

GESELLSCHAFT  
FÜR INFORMATIK





Birgitta König-Ries, Stefanie Scherzinger,  
Wolfgang Lehner, Gottfried Vossen  
(Hrsg.)

**Datenbanksysteme für  
Business, Technologie und Web  
(BTW 2023)**

**06. – 10. März 2023  
in Dresden, Deutschland**

Gesellschaft für Informatik e.V. (GI)

## **Lecture Notes in Informatics (LNI) - Proceedings**

Series of the Gesellschaft für Informatik (GI)

Volume P-331

ISBN 978-3-88579-725-8

ISSN 1617-5468

### **Volume Editors**

Prof. Dr. Birgitta König-Ries

Friedrich-Schiller-Universität Jena

Lehrstuhl für verteilte Informationssysteme

Leutragraben 1, 07743 Jena, Germany

Email: [birgitta.koenig-ries@uni-jena.de](mailto:birgitta.koenig-ries@uni-jena.de)

Prof. Dr. Stefanie Scherzinger

Universität Passau

Lehrstuhl für Informatik mit Schwerpunkt Skalierbare Datenbanksysteme

Innstraße 43, 94032 Passau, Germany

Email: [stefanie.scherzinger@uni-passau.de](mailto:stefanie.scherzinger@uni-passau.de)

Prof. Dr.-Ing. Wolfgang Lehner

Technische Universität Dresden

Lehrstuhl für Datenbanken

Nöthnitzer Straße 46, 01187 Dresden, Germany

Email: [wolfgang.lehner@tu-dresden.de](mailto:wolfgang.lehner@tu-dresden.de)

Prof. Dr. Gottfried Vossen

Universität Münster

European Research Center for Information Systems (ERCIS)

Leonardo-Campus 3, 48149 Münster, Germany

Email: [vossen@uni-muenster.de](mailto:vossen@uni-muenster.de)

### **Series Editorial Board**

Andreas Oberweis, KIT Karlsruhe,

(Chairman, andreas.oberweis@kit.edu)

Torsten Brinda, Universität Duisburg-Essen, Germany

Dieter Fellner, Technische Universität Darmstadt, Germany

Ulrich Flegel, Infineon, Germany

Ulrich Frank, Universität Duisburg-Essen, Germany

Michael Goedicke, Universität Duisburg-Essen, Germany

Ralf Hofestädt, Universität Bielefeld, Germany

Wolfgang Karl, KIT Karlsruhe, Germany

Michael Koch, Universität der Bundeswehr München, Germany

Peter Sanders, Karlsruher Institut für Technologie (KIT), Germany

Andreas Thor, HTWK Leipzig, Germany

Ingo Timm, Universität Trier, Germany

Karin Vosseberg, Hochschule Bremerhaven, Germany

Maria Wimmer, Universität Koblenz-Landau, Germany

### **Dissertations**

Rüdiger Reischuk, Universität Lübeck, Germany

### **Thematics**

Agnes Koschmider, Universität Kiel, Germany

### **Seminars**

Judith Michael, RWTH Aachen, Germany

© Gesellschaft für Informatik, Bonn 2023

printed by Köllen Druck+Verlag GmbH, Bonn



*This book is licensed under a Creative Commons BY-SA 4.0 licence.*

## Vorwort

Die 20. Fachtagung “Datenbanksysteme für Business, Technologie und Web” (BTW‘23) des Fachbereichs “Datenbanken und Informationssysteme” (DBIS) der Gesellschaft für Informatik (GI) findet vom 6. bis 10. März 2023 an der Technischen Universität Dresden statt (<https://btw2023-dresden.de>). Die BTW‘23 sticht dabei nicht nur durch das Jubiläum hervor – seit 40 Jahren wird sie alle zwei Jahre an einem anderen Standort veranstaltet und findet somit in diesem Jahr zum 20. Mal statt – sondern sie ist nach der Corona-Pandemie eine der ersten Möglichkeiten der Datenbank-Community in Deutschland und den Nachbarländern, sich wieder in Präsenz zu treffen und fachlich auszutauschen. Organisiert wird sie von der Database Research Group der TU Dresden (<https://www.db.inf.tu-dresden.de/>), die soeben ihr 20-jähriges Bestehen beging. Auch die BTW‘21 wurde vom Team der TU Dresden organisiert, musste jedoch pandemiebedingt in den virtuellen Raum verlegt werden. Im Rahmen einer wöchentlichen Lecture Series fand das wissenschaftliche Programm einschließlich der geplanten Keynotes während des Sommersemesters 2021 großen Zuspruch. Umso mehr freuen wir uns auf die 20. Ausgabe der BTW – nach 1995 – zum zweiten Mal vor Ort in der sächsischen Landeshauptstadt Dresden.

Der Großraum Dresden, bekannt für die Vielzahl touristischer Highlights, hat sich mittlerweile zu einem weltweit sichtbaren Standort der High-Tech Industrie entwickelt. Neben der Halbleiterindustrie hat sich in den letzten Jahren eine junge und extrem erfolgreiche Software-Szene etabliert, die das gesamte Portfolio von Embedded Systems (als logische Ergänzung der klassischen hardware-orientierten Industrie) bis hin zu KI-Lösungen umfasst. Neue Forschungsinstitute interagieren im Umfeld einer agilen Gründerszene. „Data as the new oil“ ist in diesem Kontext von zentralem Interesse für die Wissenschaft und Wirtschaft. Entsprechend zeugt die BTW-Schirmherrschaft des Ministerpräsidenten des Freistaates Sachsen, Herrn Michael Kretschmer, von großem Interesse und umfangreicher Unterstützung seitens der Politik.

Die BTW als Treffpunkt für die deutschsprachige Datenbank-Community hat von Anfang an Internationalität gelebt. Mittlerweile sind internationale Teilnehmende, ausschließlich englische Tagungsbeiträge, sowie Englisch als die Vortragssprache zur Normalität geworden. Auch das Spektrum der Beiträge hat sich kontinuierlich verändert. Das wissenschaftliche Kernprogramm wird ergänzt durch ein Demonstrationsprogramm und ein Studierendenprogramm, letzteres wurde erstmals auch zentral in das Hauptprogramm integriert. Die Data Science Challenge, vier Workshops, zwei Tutorials und ein eigener Industrietag („Dresden Data Day“), welcher neben Vorträgen von „Big Playern“ auch die regionale IT-Industrie involviert, komplettieren das Programm der BTW‘23.

Einreichungen zum wissenschaftlichen Hauptprogramm konnten einem oder mehreren von vier Tracks, die die fachliche Breite der Datenbank-Community konsequent abdecken, zugeordnet werden: „Database Technologies and Systems“, „Text, Semi-structured Data and IR“, „Data Engineering, Data Science and Machine Learning“ sowie „Database Languages and Theory“. Bewusst wurden Beiträge aus der Praxis in die einzelnen Tracks mit einbezogen. Die hohe Zahl von Einreichungen, bei denen mehr als ein Track angegeben wurde, spiegelt die engen Querbezüge zwischen unterschiedlichen Schwerpunkten der Datenbankforschung wider.

Aus insgesamt 60 Einreichungen wurden 19 reguläre Beiträge sowie 12 Kurzbeiträge zur Präsentation angenommen. Der eigens organisierte Review-Prozess mit einer zusätzlichen Feedback- und Revision-Phase und das hohe Engagement der PC-Mitglieder schufen hierbei beste Voraussetzungen für diese qualitativ hochwertigen Beiträge für das Tagungsprogramm.

Zum zweiten Mal wurde im Begutachtungsprozess besonderes Augenmerk auf die Reproduzierbarkeit wissenschaftlicher Artefakte, also von Experimenten und Analysen gelegt. Autorinnen und Autoren der akzeptierten Beiträge konnten ihre Arbeiten einem entsprechenden Verfahren unterziehen lassen. Die Beiträge, die dies erfolgreich durchlaufen haben, sind im vorliegenden Tagungsband entsprechend gekennzeichnet.

Als wissenschaftliche Keynote-Speaker konnten zwei herausragende Persönlichkeiten der internationalen Datenbank-Forschung gewonnen werden: Sihem Amer-Yahia, CNRS Research Director Laboratoire d'Informatique de Grenoble/Frankreich mit einem Vortrag zum Thema „Commodifying Data Exploration“ sowie Andy Pavlo, Carnegie Mellon University/USA mit seinen Ausführungen über „Why Machine Learning for Automatically Optimizing Databases Doesn't Work“.

Die „Fresh Thinking Talks“ – erstmals bei der virtuell durchgeführten BTW'21 eingeführt – setzen innovative Impulse. Meike Klettke, Universität Regensburg spricht zum Thema “Between Data Lakes and Research Data Management – Data Engineering Tasks for the Next Decade”. Der Beitrag von Wolfram Wingerath, Universität Oldenburg steht unter dem Motto „What You Say is What You Get: Hands-Free Coding in 2023“.

Ergänzt wird das wissenschaftliche BTW-Programm durch im Vorfeld stattfindende Workshops, in denen aktuelle Trends aufgegriffen werden und im kleinen Rahmen anhand von Impulsvorträgen diskutiert werden. Diese Beiträge sind auch in diesem Tagungsband enthalten:

- Workshop on Novel Data Management Ideas on Heterogeneous Hardware Architectures (NoDMC)
- Workshop on Big (and Small) Data in Science and Humanities (BigDS2023)
- Workshop on Data Engineering for Data Science (DE4DS)
- A Tutorial Workshop on ML for Systems and Systems for ML

Zwei Tutorials, die jeweils parallel zum wissenschaftlichen Programm abgehalten werden, ergänzen das Programmportfolio der BTW'23. Mit dem Tutorial “From BERT to GPT-3 Codex: Harnessing the Potential of Very Large Language Models for Data Management” adressiert Immanuel Trummer, Cornell University/USA einen extrem spannenden und für die Rolle von Datenmanagement in modernen Anwendungsszenarien überaus relevanten Themenkomplex. Auf der Systemebene wird das Tutorienprogramm durch ein von Alberto Lerner, University of Fribourg/Schweiz und Philippe Bonnet, ITU University of Copenhagen/Dänemark gemeinsam angebotenes Tutorial zum Thema „The Principles of Database and SSDs Co-Design” getragen.

Das Demo-Programm bietet die Möglichkeit, praktische Ergebnisse der Forschung im Bereich der Datenbank- und Informationssystemtechnologien auf interaktive Weise vorzustellen. Von den 11 Einreichungen wurden sechs Beiträge akzeptiert, die während der Konferenz präsentiert werden und deren Beschreibungen in diesem Tagungsband veröffentlicht sind.

Die Förderung des wissenschaftlichen Nachwuchses ist ein zentrales Anliegen des Fachbereichs DBIS. Das Studierendenprogramm der BTW ermöglicht Studierenden und angehenden Nachwuchswissenschaftlerinnen und Nachwuchswissenschaftlern, ihre Arbeiten einem Fachpublikum vorzustellen, das Fachgebiet "Datenbanken und Informationssysteme" kennenzulernen und sich mit Fachleuten und Gleichgesinnten aus Wissenschaft und Praxis auszutauschen und zu vernetzen. Erstmals wird bei der BTW23 das Studierendenprogramm in das Hauptprogramm integriert: so haben die Studierenden freien Zugang zu allen Programm-Elementen der BTW, können ihre Arbeiten in den Poster-Sessions präsentieren und erhalten ein gezieltes Mentoring durch wissenschaftliche Persönlichkeiten der Datenbank-Community. Selbstverständlich werden die qualitätsgesicherten Beiträge des Studierendenprogramms auch in diesem vorliegenden Tagungsband veröffentlicht. Zum 12. Mal werden bei der BTW mit dem Dissertationspreis des GI-Fachbereichs DBIS hervorragende Dissertationen aus dem Gebiet der Konzepte, Verfahren, Erweiterungen und Anwendungen von Datenmanagement- und Informationssystemen gewürdigt. Aus den insgesamt zwölf Einreichungen wurden zwei Arbeiten ausgezeichnet und Kurzfassungen in diesem Tagungsband veröffentlicht: „Adaptive Architectures for Robust Database Management Systems“, Dissertationsschrift von Tiemo Bang an der TU Darmstadt und „Enhancing Explainability and „Scrutability of Recommender Systems“ Dissertationsschrift von Azin Ghazimatin an der Universität des Saarlands.

Neben dem Dissertationspreis werden weitere Awards im Rahmen der BTW'23 verliehen: Mit dem „Best Long Paper Award“ für den besten wissenschaftlichen Beitrag wird die Arbeit mit dem Titel „WannaDB: Ad-hoc SQL Queries over Text Collections“ von Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban und Carsten Binnig ausgezeichnet. Mit dem „Best Short Paper Award“ für den besten wissenschaftlichen Kurzbeitrag wird die Arbeit zum Thema „NN2SQL: Let SQL Think for Neural Networks“ von Maximilian Emanuel Schüle, Alfons Kemper und Thomas Neumann ausgezeichnet. Des Weiteren wird ein „Best Demo Award“ für die innovativste Demo vergeben.

Im Rahmen der Data Science Challenge haben sich vier teilnehmende Gruppen der Herausforderung der Lösung konkreter datenzentrierter Anwendungsproblemen gestellt. Thema der im Rahmen der BTW'23 stattfindenden Data Science Challenge sind städtische Verkehrs- und Fahrraddaten und die Entwicklung innovativer Konzepte für die Verkehrswende von morgen.

Erstmals auf der BTW'23 wurde das Industrieprogramm ausgegliedert und zu einem eigenständigen „Industrietag“ erweitert (<http://dresden-data-day.de/>). So steht der Freitag als abschließender Konferenztag im Zeichen der Vernetzung mit der Praxis durch eine Keynote von Stefan Bäuerle (SAP SE) zum Thema „Enabling Enterprise Data Applications with SAP HANA Cloud“, Sessions mit eingeladenen Vorträgen der regionalen und überregionalen Industrie sowie Break-Out Sessions. Als Co-Organisator konnte hierfür der Silicon Saxony e.V., die Digitalagentur Sachsen sowie als idealer Gastgeber die Sächsische Aufbaubank SAB als Partner für die BTW'23 gewonnen werden.

Ein ganz besonderer Dank geht natürlich an die Vielzahl der Sponsoren, ohne deren Engagement die Durchführung einer BTW nicht möglich wäre. Auch gilt ein großer Dank der GI-Geschäftsstelle für die umfangreiche Unterstützung in der finanziellen Abwicklung der Tagung.

Vielen Dank an alle Beteiligten für das große Engagement!  
Dresden, im Februar 2023

## Sponsoren

Wir danken den folgenden Unternehmen für die Unterstützung der Konferenz.

### Platinum



### Gold



### Silber



### Bronze



### **Tagungsleitung / General Chair**

Wolfgang Lehner, TU Dresden

### **Organisationsteam der BTW'23**

Ulrike Schöbel, TU Dresden

Dirk Habich, TU Dresden

Maik Thiele, HTW Dresden

### **Ehrevorsitzender und Organisation Dissertationspreis**

Gottfried Vossen, Universität Münster

### **PC Chairs**

Birgitta König-Ries, Universität Jena

Stefanie Scherzinger, Universität Passau

### **Track Chairs**

Meike Klettke, Universität Regensburg

Viktor Leis, TU München

Wim Martens, Universität Bayreuth

Philipp Schaer, TH Köln

### **Gutachterinnen und Gutachter wissenschaftliches Hauptprogramm**

Christian Beecks, Fernuniversität in Hagen

Klaus Berberich, HTW Saarland

Christoph Berkholz, HU Berlin

Carsten Binnig, TU Darmstadt

Alexander Bondarenko, Universität Jena

Timo Breuer, TH Köln

Leyla Jael Castro, ZB MED

Jens Dittrich, Universität des Saarlandes

Michael Elberfeld, THM Gießen

George Fletcher, TU Eindhoven

Dominik Freydenberger, Loughborough University

Rainer Gemulla, Universität Mannheim

Michael Gertz, Universität Heidelberg

Anika Groß, Hochschule Anhalt

Torsten Grust, Universität Tübingen

Claudia Hauff, TU Delft

Gernot Heisenberg, TH Köln

Andreas Henrich, Universität Bamberg

Katja Hose, Aalborg University

Alfons Kemper, TU München

Dagmar Kern, GESIS

Udo Kruschwitz, Universität Regensburg

Ulf Leser, HU Berlin

Thomas Mandl, Universität Hildesheim

Norman May, SAP

Philipp Mayr-Schlegel, GESIS  
Stefan Mengel, CNRS  
Jelena Mitrovic, Universität Passau  
Ingo Müller, Google Zürich  
Felix Naumann, Hasso Plattner Institut Potsdam  
Thomas Neumann, TU München  
Daniela Nicklas, Universität Bamberg  
Thorsten Papenbrock, Philipps-Universität Marburg  
Gunter Saake, Universität Magdeburg  
Ralf Schenkel, Universität Trier  
Felix Schuhknecht, Universität Mainz  
Maximilian E. Schüle, Universität Bamberg  
Bernhard Seeger, Philipps-Universität Marburg  
Thomas Seidl, LMU München  
Christin Seifert, Universität Duisburg-Essen  
Günther Specht, Universität Innsbruck  
Knut Stolze, IBM  
Uta Störl, Fernuniversität in Hagen  
Andreas Thor, HTWK Leipzig  
Nils Vortmeier, Universität Zürich  
Lena Wiese, Goethe-Universität Frankfurt  
Wolfram Wingerath, Universität Oldenburg  
Tobias Ziegler, TU Darmstadt

### **Demo Program**

**Vorsitz:** Tilmann Rabl, Hasso Plattner Institut Potsdam

David Broneske, Deutsches Zentrum für Hochschul- und Wissenschaftsforschung GmbH

Hazar Harmouch, Hasso Plattner Institut Potsdam

Alexander Krause, TU Dresden

Ruben Mayer, TU München

Ingo Müller, Google Zurich

Eleni Tzirita Zacharitou, IT University Copenhagen

### **Workshops and Tutorials**

**Vorsitz:** Carsten Binnig, TU Darmstadt

### **Studierendenprogramm**

**Vorsitz:** Rainer Gemulla, Universität Mannheim

Harald Kosch, Universität Passau

Sebastian Michel, TU Kaiserslautern

Thomas Rakow, Hochschule Düsseldorf

Eike Schallehn, Universität Magdeburg

Uta Störl, Fernuniversität in Hagen

Jens Teubner, TU Dortmund

Andreas Thor, HTWK Leipzig

**Reproducibility Committee:**

**Vorsitz:** Ziawasch Abedjan, Leibniz Universität Hannover  
Sheeba Samuel, Universität Jena

Matthias Böhm, TU Berlin

Paul Blockhaus, Otto-von-Guericke-Universität Magdeburg

Pascal Hirmer, Universität Stuttgart

Peter Reimann, Universität Stuttgart

Felix Schuhknecht, Universität Mainz

Mohammed Sedir, Hasso Plattner Institut Potsdam

Christoph Stach, Universität Stuttgart

Ahmed Waqas, Friedrich-Schiller-Universität Jena

**Proceedings Chair**

Alexander Krause, TU Dresden

**Data Science Challenge**

Eric Peukert, Universität Leipzig

# Contents

## Scientific Program

### Session 1

**Goetz Graefe**

*Priority queues for database query processing* . . . . . 27

**Ben Hurdelhey, Marcel Weisgut, Martin Boissier**

*Workload-Driven Data Placement for Tierless In-Memory Database Systems* 47

**Kevin Wellenzohn, Michael H. Böhlen, Sven Helmer, Marcel Reutegger**

*Workload-Aware Contention-Management in Indexes for Hierarchical Data* 71

**Florian Eppinger, Uta Störl**

*Tuning Cassandra through Machine Learning* . . . . . 93

**Muhammad Attahir Jibril, Alexander Baumstark, Kai-Uwe Sattler**

*GTPC: Towards a Hybrid OLTP-OLAP Graph Benchmark* . . . . . 105

**Knut Stolze, Felix Beier, Vassil Dimov, Eirini Kalogeiton, Mateo Tošić**

*IBM Data Gate: Making On-Premises Mainframe Databases Available to Cloud Applications* . . . . . 119

**Felix Schuhknecht, Simon Jörz**

*The Easiest Way of Turning your Relational Database into a Blockchain — and the Cost of Doing So* . . . . . 131

### Session 2

**Benjamin Hättasch, Jan-Micha Bodensohn, Liane Vogel, Matthias Urban, Carsten Binnig**

*WannaDB: Ad-hoc SQL Queries over Text Collections* . . . . . 157

|  |     |
|--|-----|
| <b>Maximilian E. Schüle, Alfons Kemper, Thomas Neumann</b><br><i>NN2SQL: Let SQL Think for Neural Networks</i> . . . . .   | 183 |
| <b>Dennis Aumiller, Jing Fan, Michael Gertz</b><br><i>On the State of German (Abstractive) Text Summarization</i> . . . . .  | 195 |
| <b>Manfred Moosleitner, Günther Specht, Eva Zangerle</b><br><i>Detection of Generated Text Reviews by Leveraging Methods from<br/>Authorship Attribution: Predictive Performance vs. Resourcefulness</i> . . . | 221 |

### Session 3

|  |     |
|--|-----|
| <b>Alice Rey, Michael Freitag, Thomas Neumann</b><br><i>Seamless Integration of Parquet Files into Data Processing</i> . . . . .   | 235 |
| <b>Adnan Alhomssi, Michael Haubenschild, Viktor Leis</b><br><i>The Evolution of LeanStore</i> . . . . .  | 259 |
| <b>Rico Bergmann, Axel Hertzschuch, Claudio Hartmann, Dirk Habich,<br/>Wolfgang Lehner</b><br><i>PostBOUND: PostgreSQL with Upper Bound SPJ Query Optimization</i> . .   | 283 |
| <b>Ivan Ilic, Ilin Tolovski, Tilmann Rabl</b><br><i>RMG Sort: Radix-Partitioning-Based Multi-GPU Sorting</i> . . . . .   | 305 |
| <b>Dennis Treder-Tschechlov, Peter Reimann, Holger Schwarz, Bernhard<br/>Mitschang</b><br><i>Approach to Synthetic Data Generation for Imbalanced Multi-class<br/>Problems with Heterogeneous Groups</i> . . . . . | 329 |
| <b>John Ziegler, Michael Gertz</b><br><i>No Mayfly: Detection and Analysis of Long-term Twitter Trends</i> . . . . .   | 353 |

### Session 4

|   |     |
|---|-----|
| <b>Maximilian Koch, Mahdi Esmailoghli, Sören Auer, Ziawasch Abedjan</b><br><i>Duplicate Table Discovery with Xash</i> . . . . . | 367 |
|---|-----|

|   |     |
|---|-----|
| <b>Marcian Seeger, Sebastian Schmidl, Alexander Vielhauer, Thorsten Papenbrock</b><br><i>DPQL: The Data Profiling Query Language . . . . .</i>  | 391 |
| <b>Leonardo Hübscher, Lan Jiang, Felix Naumann</b><br><i>ExtracTable: Extracting Tables from Raw Data Files . . . . .</i>   | 417 |
| <b>Florens Rohde, Martin Franke, Victor Christen, Erhard Rahm</b><br><i>Value-specific Weighting for Record-level Encodings in<br/>Privacy-Preserving Record Linkage . . . . .</i>                                    | 439 |
| <b>Sebastian Schmidl, Phillip Wenig, Thorsten Papenbrock</b><br><i>HYPEX: Hyperparameter Optimization in Time Series Anomaly Detection</i>  | 461 |
| <b>Christopher Rost, Kevin Gomez, Peter Christen, Erhard Rahm</b><br><i>Evolution of Degree Metrics in Large Temporal Graphs . . . . .</i>  | 485 |
| <br>Session 5   |     |
| <b>Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, Matthias Weidlich</b><br><i>Discovering Multi-Dimensional Subsequence Queries from Traces – From<br/>Theory to Practice . . . . .</i> | 511 |
| <b>Henriette Behr, Volker Markl, Zoi Kaoudi</b><br><i>Learn What Really Matters: A Learning-to-Rank Approach for ML-based<br/>Query Optimization . . . . .</i>  | 535 |
| <b>Maximilian Mayerl, Michael Vötter, Günther Specht, Eva Zangerle</b><br><i>Pairwise Learning to Rank for Hit Song Prediction . . . . .</i>  | 555 |
| <b>Christian Winter, Moritz Sichert, Altan Birler, Thomas Neumann,<br/>Alfons Kemper</b><br><i>Communication-Optimal Parallel Reservoir Sampling . . . . .</i>  | 567 |
| <b>Philipp Christmann, Rishiraj Saha Roy, Gerhard Weikum</b><br><i>CLOCQ: A Toolkit for Fast and Easy Access to Knowledge Bases . . . . .</i>   | 579 |

## Session 6

|   |     |
|---|-----|
| <b>Manh Khoi Duong, Jannik Dunkelau, José Andrés Cordova, Stefan Conrad</b><br><i>RAPP: A Responsible Academic Performance Prediction Tool for Decision-Making in Educational Institutes</i> . . . . .      | 595 |
| <b>Stefan Brass, Alexander Hinneburg</b><br><i>Semantic Watermarks for Detecting Cheating in Online Database Exams</i> .  | 607 |
| <b>Thomas C. Rakow, André Kless, Charlotte Hasler, Harm Knolle, Heide Faeskorn-Woyke, Inga Marina Saatz, Jens Lambert, Mareike Focken</b><br><i>Developing OERs for Teaching Database Systems</i> . . . . . | 621 |
| <b>Azin Ghazimatin</b><br><i>Enhancing Explainability and Scrutability of Recommender Systems</i> . . .   | 633 |
| <b>Tiemo Bang</b><br><i>Adaptive Architectures for Robust Data Management Systems</i> . . . . .   | 641 |
| <br>Demo Track  |     |
| <b>Lucas Woltmann, Katja Ferger, Claudio Hartmann, Wolfgang Lehner</b><br><i>JumpXClass: Explainable AI for Jump Classification in Trampoline Sports</i>  | 651 |
| <b>Mirjam Bayer, Yorik Timo Hansen, Kimberley Kosbü, Andrea Kulow, Peer Kröger</b><br><i>UniDash: Interactive Dashboard for Data Driven Insights on Universities</i>  | 657 |
| <b>Alexander Kerth, Felix Schuhknecht, Lukas Pensel, Justus Henneberg</b><br><i>Better Safe than Sorry: Visualizing, Predicting, and Successfully Guiding Courses of Study</i> . . . . .                    | 665 |
| <b>Eric Tröbs, Stefan Hagedorn, Kai-Uwe Sattler</b><br><i>JPTest - Grading Data Science Exercises in Jupyter Made Short, Fast and Scalable</i> . . . . .  | 673 |
| <b>Joshua Reibert, Arne Osterthun, Marcus Paradies</b><br><i>MEDUSE: Interactive and Visual Exploration of Ionospheric Data</i> . . . . .   | 681 |

**Johannes Schildgen, Florian Heinz**  
*Interactive SQL Queries and Program Code in Presentations* . . . . . 687

Workshop Track

**Dirk Habich, David Brioneske**  
*Second Workshop on Novel Data Management Ideas on Heterogeneous  
(Co-)Processors (NoDMC)* . . . . . 697

**Andreas Henrich, Naouel Karam, Birgitta König-Ries, Bernhard  
Seeger**  
*Fourth Workshop on Big (and Small) Data in Science and Humanities (BigDS)* 701

**Ralf Schenkel, Ansgar Scherp**  
*Workshop on Data Engineering for Data Science (DE4DS)* . . . . . 705

**Manisha Luthra, Andreas Kipf, Matthias Boehm**  
*A Tutorial Workshop on ML for Systems and Systems for ML* . . . . . 707

Workshop on Novel Data Management Ideas on Heterogeneous Hardware  
Architectures (NoDMC)

**Adrian Lutsch, Gagandeep Singh, Martin Mundt, Ragnar Mogk,  
Carsten Binnig**  
*Benchmarking the Second Generation of Intel SGX for Machine Learning  
Workloads* . . . . . 711

**Felix Schuhknecht, Tamjidul Islam**  
*Inter-Query Parallelism on Heterogeneous Multi-Core CPUs* . . . . . 719

**Tobias Hahn, Daniel Schüll, Stefan Wildermann, Jürgen Teich**  
*An FPGA Avro Parser Generator for Accelerated Data Stream Processing* 729

**Andreas Geyer, Daniel Ritter, Dong Hun Lee, Minseon Ahn, Johannes  
Pietrzyk, Alexander Krause, Dirk Habich, Wolfgang Lehner**  
*Working with Disaggregated Systems. What are the Challenges and  
Opportunities of RDMA and CXL?* . . . . . 751

|  |     |
|--|-----|
| <b>Lawrence Benson, Marcel Weisgut, Tilmann Rabl</b><br><i>What We Can Learn from Persistent Memory for CXL</i> . . . . .  | 757 |
| <b>Johannes Fett, Christian Schwarz, Urs Kober, Dirk Habich, Wolfgang Lehner</b><br><i>Improving GPU Matrix Multiplication by Leveraging Bit Level Granularity and Compression</i> . . . . .                                 | 763 |
| <b>Alex El-Shaikh, Bernhard Seeger</b><br><i>DNAContainer: An object-based storage architecture on DNA</i> . . . . .   | 773 |
| <b>Alexander Baumstark, Muhammad Attahir Jibril, Kai-Uwe Sattler</b><br><i>Accelerating Large Table Scan using Processing-In-Memory Technology</i> . .   | 797 |
| <b>Patrick Damme, Matthias Boehm</b><br><i>Enabling Integrated Data Analysis Pipelines on Heterogeneous Hardware through Holistic Extensibility</i> . . . . .  | 815 |
| <br>Workshop on Big (and Small) Data in Science and Humanities (BigDS)   |     |
| <b>Johannes Schildgen, Florian Heinz, Andreas Olijnyk, Arvid Lindenau</b><br><i>Using SQL/MED to Query Heterogeneous Data Sources with Alexa Voice Commands</i> . . . . .  | 821 |
| <b>Robin Jegan, Leon Fruth, Tobias Gradl, Andreas Henrich</b><br><i>Integrating Access to Authority Data for Improved Interoperability of Research Data in the Digital Humanities</i> . . . . .                              | 829 |
| <b>Christian Beilschmidt, Johannes Dröner, Michael Mattig, Philip Schweitzer, Bernhard Seeger</b><br><i>Geo Engine: Workflow-backed Geo Data Portals</i> . . . . .   | 837 |
| <b>Felicitas Löffler, Fateme Shafiei, René Witte, Birgitta König-Ries, Friederike Klan</b><br><i>Semantic Search for Biological Datasets: A Usability Study on Modes of Querying and Explaining Search Results</i> . . . . . | 851 |
| <b>Wolfgang Müller, Lukrécia Mertová</b><br><i>ReStoRunT: Simple Recording, Storing, Running and Tracing changes in Spreadsheets</i> . . . . .   | 865 |

|   |     |
|---|-----|
| <b>Aly Abdelmageed, Shahenda Hatem, Tasneem Wael, Walaa Medhat, Birgitta König-Ries, Susan F. Ellakwa, Passent Elkafrawy, Alsayed Algergawy</b><br><i>A Core Ontology to Support Agricultural Data Interoperability . . . . .</i> | 879 |
| <b>Elena Volkanovska, Sherry Tan, Changxu Duan, Sabine Bartsch and Wolfgang Stille</b><br><i>The InsightsNet Climate Change Corpus (ICCC) . . . . .</i>   | 887 |
| <b>Sebastian Bruchhaus, Thoralf Reis, Marco Xaver Bornschlegl, Uta Störl, Matthias Hemmje</b><br><i>Towards a User-Empowering Architecture for Trustability Analytics . . . . .</i>   | 901 |
| <br>Workshop on Data Engineering for Data Science (DE4DS)   |     |
| <b>Valerie Restat, Meike Klettke, Uta Störl</b><br><i>“FAIR ” is not enough – A Metrics Framework to ensure Data Quality through Data Preparation . . . . .</i>   | 917 |
| <b>Maximilian E. Schüle</b><br><i>Recursive SQL and GPU-Support for In-Database Machine Learning . . . . .</i>  | 931 |
| <b>Sabrina Göllner, Marina Tropmann-Frick</b><br><i>VERIFAI - A Step Towards Evaluating the Responsibility of AI-Systems . . . . .</i>  | 933 |
| <b>David Burrell, Xenofon Chatziliadis, Eleni Tziritza Zacharatou, Steffen Zeuch, Volker Markl</b><br><i>Workload Prediction for IoT Data Management Systems . . . . .</i>  | 943 |
| <b>Erik Kleinsteuber, Samira Babalou, Birgitta König-Ries</b><br><i>A Provenance Management Framework for Knowledge Graph Generation in a Web Portal . . . . .</i>  | 951 |
| <b>Dominik Kerzel, Birgitta König-Ries, Sheeba Samuel</b><br><i>MLProvLab: Provenance Management for Data Science Notebooks . . . . .</i>   | 965 |
| <b>Pronaya Prosun Das, Marcel Mast, Lena Wiese, Thomas Jack, Antje Wulff</b><br><i>Data Extraction for Associative Classification using Mined Rules in Pediatric Intensive Care Data . . . . .</i>                                | 981 |

|   |      |
|---|------|
| <b>Sven Langenecker, Christoph Sturm, Christian Schalles, Carsten Binnig</b>  |      |
| <i>SportsTables: A new Corpus for Semantic Type Detection . . . . .</i>   | 995  |
| <b>Björn Engelmann, Philipp Schaer</b>  |      |
| <i>Reliable Rules for Relation Extraction in a Multimodal Setting . . . . .</i>   | 1009 |
| <b>Arne Grünhagen, Marina Tropmann-Frick, Annika Eichler, Görschwin Fey</b>   |      |
| <i>Predictive Maintenance for the Optical Synchronization System of the European XFEL: A Systematic Literature Survey . . . . .</i> | 1023 |
| <br>Student Track   |      |
| <b>Jüri Keller, Meik Bittkowski, Philipp Schaer</b>   |      |
| <i>Automated Statement Extraction from Press Briefings . . . . .</i>  | 1049 |
| <b>Tarek Al Mustafa, Birgitta König-Ries, Sheeba Samuel</b>   |      |
| <i>MLProvCodeGen: A Tool for Provenance Data Input and Capture of Customizable Machine Learning Scripts . . . . .</i>               | 1059 |
| <b>Tim Fischer</b>  |      |
| <i>To Iterate Is Human, to Recurse Is Divine — Mapping Iterative Python to Recursive SQL . . . . .</i>                              | 1069 |
| <b>Jerome Thiessat, Lucas Woltmann, Claudio Hartmann, Dirk Habich</b>   |      |
| <i>Optimizing Query Processing in PostgreSQL Through Learned Optimizer Hints . . . . .</i>  | 1075 |
| <b>Lucas Fabian Naumann</b>   |      |
| <i>WEB TENSOR: Towards high-performance raster data analysis in the browser</i>   | 1083 |
| <b>Tim Gutberlet, Janik Sauerbier</b>   |      |
| <i>Which Rules Entail this Fact? - An Efficient Approach Using RDBMSs . . .</i>   | 1091 |
| <b>Lukas Laskowski, Florian Sold</b>  |      |
| <i>Explainable Data Matching: Selecting Representative Pairs with Active Learning Pair-Selection Strategies . . . . .</i>           | 1099 |

|  |      |
|--|------|
| <b>Benjamin Uwe Killisch, Thomas Kudraß, Florian Scheffler</b><br><i>Efficient handling of recursive relationships in ORM frameworks using<br/>Entity Framework Core as an example . . . . .</i> | 1105 |
| <b>Christoph Köhnen</b><br><i>Witness Generation for JSON Schema Patterns . . . . .</i>  | 1113 |

## List of Authors



# Scientific Program



# Session 1



# Priority queues for database query processing

## New techniques for tree-of-losers priority queues and for offset-value coding

Goetz Graefe<sup>1</sup>

**Abstract:** Interesting orderings let sort-based query processing out-perform hash-based algorithms, but only tree-of-losers priority queues and offset-value coding permit competing in all cases including large unsorted inputs with large or complex keys. As long as this competition persists, alternative algorithms with equivalent functionality will plague query execution, e.g., in software maintenance and in query plan scheduling, and mistaken algorithm choices will plague query optimization, e.g., for joins, intersection, and grouping.

After explaining tree-of-losers priority queues and offset-value coding, our work introduces necessary extensions for efficient run generation (in external merge sort) with variable-size records. The required changes in tree-of-losers priority queues support increasing and decreasing any key value at any time in logarithmic time, including incremental maintenance of offset-value codes, with the expected time for key value increases independent of the size of the priority queue. As all kinds of scheduling applications use priority queues, our contributions go beyond database query processing. A discussion of double-ended priority queues illustrates the concepts.

This may be the first time that tree-of-losers priority queues are extended to addressable priority queues and to non-monotone sequences of input keys; and that offset-value coding is extended to non-monotone sequences of input keys. The proposed solutions and the included code snippets are simple, small, and fast, in contrast to the time and effort spent on bringing them to this state.

**Keywords:** sorting; grouping; merge join; interesting orderings; tree of losers; offset-value coding.

## 1 Introduction

In many classic algorithms for database query execution [BE77, Ep79], from in-stream duplicate removal, grouping, and aggregation (for sorted streams) to in-sort grouping, merge join, and index nested-loops join, each algorithm's core is either a database index or a sort operation. Hash-based query execution algorithms, e.g., [Br84, DG85, De84, KTM83, NKT88], seem to have changed that, but other than computing hash values from column values, hash join and hash aggregation have at their core an index, i.e., a hash table, and a sort, i.e., internal and external distribution sort [IS56] on hash values rather than column values. Thus, sorting techniques are crucial for efficient database query processing, e.g., distribution sort, quicksort [Ho62], merge sort [Fr56], and priority queues.

Tree-of-losers priority queues [Go63, Kn98] are more efficient than the traditional and better known tree-of-winners priority queues because the former use only leaf-to-root

---

<sup>1</sup> Google; Madison, Wisconsin, USA GoetzG@Google.com

passes with  $\log_2 N$  comparisons, whereas the latter also need root-to-leaf passes with up to  $2 \times \log_2 N$  comparisons. Therefore, tree-of-losers priority queues can guarantee internal and external sorting with counts of comparisons practically equal to the provable lower bound<sup>2</sup>. Just as importantly, tree-of-losers priority queues work with offset-value coding [Co77], which minimizes the effort per row comparison by combining prefix truncation and order-preserving surrogate keys. Together, these techniques guarantee at most  $N \times K$  column value comparisons when sorting  $N$  rows with  $K$  key columns; the remainder of the required  $\log_2(N!)$  row comparisons are compiled-in single-instruction integer comparisons and thus similar to hash values in hash-based query processing.

Tree-of-losers priority queues seem to require monotone (ever-increasing) sequences of keys, making tree-of-losers priority queues perfect for merging sorted runs. They can be adapted [Go63] to internal sorting, run generation in read-sort-write cycles, and run generation with continuous replacement selection for fixed-size data records. In continuous replacement selection for variable-size data records, e.g., using best-fit or first-fit memory management [LG98], occupancy counts in a priority queue change frequently, which published methods for tree-of-losers priority queues cannot accommodate.

Tree-of-losers priority queues as used to-date, i.e., with complete leaf-to-root passes, can serve neither as non-monotone priority queues nor as addressable priority queues. On the other hand, only tree-of-losers priority queues can sort with comparison counts near the provable lower bound – neither quicksort nor tree-of-winners priority queues do. Moreover, only tree-of-losers priority queues can use offset-value coding for linear costs for column value comparisons – neither quicksort nor tree-of-winners priority queues do.

Initially motivated by variable-size records in replacement selection, our contributions are:

1. a small extension to leaf-to-root passes in tree-of-losers priority queues that elevates them to addressable priority queues;
2. a significant extension to leaf-to-root passes in tree-of-losers priority queues that supports non-monotone sequences of input keys, including early and late fences for invalid (not occupied) slots;
3. new techniques for offset-value coding in tree-of-losers priority queues with non-monotone sequences of input keys;
4. an implementation of double-ended priority queues that uses two priority queues in opposite sort order and that, after popping the top element from one priority queue, repairs the other one in constant expected time (independent of the size, capacity, or tree height of the priority queue); and

---

<sup>2</sup> Sorting is equivalent to finding a permutation:  $N$  distinct key values permit  $N!$  permutations; at best, each key comparison disproves half of the remaining possibilities; determining the permutation of the input requires at least  $\log_2(N!) \approx N \times \log_2(N/e)$  comparisons with Euler's number  $e \approx 19/7$ .

5. an implementation of run generation using offset-value coding and continuous replacement selection for variable-size records.<sup>3</sup>

The next section provides technical background and reviews related prior work. Section 3 introduces a small extension in leaf-to-root passes of tree-of-losers priority queues that elevates them to addressable priority queues, whereupon Section 4 introduces a significant extension to support non-monotone sequences of input keys. Section 5 employs these extensions for a double-ended priority queue using two priority queues, one ascending and one descending. Section 6 adds a third extension for efficient incremental maintenance of offset-value codes in spite of non-monotone sequences of input keys. Section 7 uses all three extensions for run generation with variable record counts in memory. The final section sums up and concludes with thoughts on adopting offset-value coding in a broader context, specifically in all sort-based algorithms for database query evaluation.

## 2 Background and related prior work

This section provides some technical background about priority queues, offset-value coding, and external merge sort.

### 2.1 Priority queues

A priority queue manages a set of pairs, each a priority and some associated information. The priority is the sort key within the pair. The associated information can be detailed information, a pointer, an array index, or something else. The purpose of priority queues is to provide the minimum (or maximum) key value very quickly (from the root of a tree) and to absorb additional key-value pairs efficiently. For simplicity, all data structures considered here for priority queues assume a balanced binary tree of height  $n$  and capacity  $2^n$  or  $2^n - 1$ .

In a tree-of-winners priority queue [Kn98], the principal invariant is that a parent's key is lower than the key values in its two children (assuming an ascending ordering of key values). Popping (removing) the key-value pair with the minimum key value moves the right-most leaf entry to the root and then pushes it into the tree (with  $2n$  comparisons in a worst-case root-to-leaf pass). A subsequent insertion requires only a leaf-to-root pass with  $n$  comparisons. Alternatively, if the removal leaves an invalid entry in the root (with logical

<sup>3</sup> Prior work [LG98] used a very early version of this technique, without explicit mention, without detail or explanation, and without offset-value coding. Early versions built a stack of move targets, which nominally cut moves to one third but on superscalar CPUs performs no better than swapping (see line 7 in Figure 1), separated partial and complete leaf-to-root traversals and their different looping conditions (see line 15 in Figure 3), gated the repair loop with an extra key comparison (see line 31 in Figure 4), supported incomplete binary trees (capacities other than  $2^n$ ), and organized the tree as b-tree of cache lines, with no benefit if the entire tournament tree easily fits into the L1 cache as recommended for sorting.

key value  $-\infty$ ), pushing (inserting) a new key value simply replaces the invalid entry but then repairs the tree invariants, which usually requires two comparisons per tree level and may require all  $n$  tree levels, for  $2n$  comparisons in a worst-case root-to-leaf pass.

### 2.1.1 Tree-of-losers priority queues

A tree-of-losers priority queue [Go63, Kn98], also known as a tournament tree, is a balanced binary tree. When mapped to an array, the tree's unary root is in array slot 0. It is efficient due to leaf-to-root passes with one comparison per tree level; root-to-leaf passes with two comparisons per tree level are not required. A pair of “pop” and “push” operations requires only a single leaf-to-root pass. The principal rules are that (i) two candidate keys compete at each node in the tree and (ii) after a comparison of two candidates, the loser remains in the node and the winner becomes a candidate in the next tree level. Thus, a new overall winner reaches the root node after  $n$  comparisons in a priority queue with  $2^n$  entries. When merging, a fixed pair of runs competes at each leaf node. Run generation using read-sort-write cycles merges “sorted runs” of a single row each. Run generation by continuous replacement selection tries to extract longer sorted runs from the unsorted input.

```

1. void PQ::pass (Index const index, Key const key)
2. {
3.     Node candidate (index, key);
4.     Index slot;
5.     for (leaf (index, slot); parent (slot), slot != root (); )
6.         if (heap [slot].less (candidate))
7.             heap [slot].swap (candidate);
8.     heap [root ()] = candidate;
9. }
```

Fig. 1: The traditional leaf-to-root pass

Figure 1 shows code extracted from a working prototype of a tree-of-losers priority queue. The “index” parameter is the information associated with the key value, e.g., a run identifier between 0 and  $F - 1$  during a merge step with fan-in  $F$ . Line 3 creates a new tree node that will be swapped into the array “heap” that holds the priority queue. Line 4 defines an index for this array; line 5 initializes it, halves it to navigate from a child to its parent, and terminates the loop at the root. An equivalent to lines 4 and 5 is “for (Index slot = capacity + index; (slot /= 2) != 0; )”, but the syntax in Figure 1 more readily enables the extensions required later. Another alternative is “for (Index slot = capacity/2 + index/2; slot != 0; slot /= 2)”, which more directly shows skipping over the non-leaf nodes in the tree and assigning two index values to each tree leaf. Lines 6 and 7 in Figure 1 compare key values and ensure that the loser remains behind as the winner becomes a candidate at the parent. Line 8 saves the overall winner in the tree's root node.

With only leaf-to-root passes (and no root-to-leaf passes), run generation and merging with tree-of-losers priority queues guarantee near-optimal comparison counts. The count

of comparisons in the best case, the worst case, and the expected case are all within rounding errors of the provable lower bound of  $\log_2(N!)$ , i.e., better than the expected case of quicksort and far better than the worst case of quicksort. This includes the expected case for replacement selection, where one additional comparison per input row doubles the expected run size, cuts the run count in half, and saves one comparison per row in the merge process. With excellent expected and worst-case run-time complexity yet very limited implementation complexity, some hardware supports tree-of-losers priority queues. More specifically, the UPT “update tree” instruction of IBM’s 370- and z-series mainframes [IB88, Iy05] implements essentially the logic of Figure 1.

In a tree-of-losers priority queue, each key value is paired with an index in the range 0 to  $2^n - 1$  for a priority queue with tree height  $n$ . During a merge step, these indexes identify runs; during run generation, they identify rows in memory or more precisely slots in an array. These indexes determine where a leaf-to-root pass starts. At all times, each index value exists exactly once in the priority queue, possibly marked as an invalid entry by a fence with logical key value  $-\infty$  or  $+\infty$ .

### 2.1.2 Addressable priority queues

In a tree-of-losers priority queue, an index value maps to a specific leaf and therefore to a specific leaf-to-root path, which can be used to identify, find, read, and perhaps modify or even delete any entry. For the same functionality, a tree-of-winners priority queue requires an additional data structure to find an entry in the tree. While this data structure permits finding an entry quickly, it must track every movement in the tree representing the priority queue, thus increasing the cost of any movement within the priority queue.

If a key value is modified or deleted, the priority queue and its invariants need repair. For tree-of-losers priority queues, Section 3 introduces efficient techniques without any additional data structure yet with efficient maintenance for key change and deletion.

### 2.1.3 Monotone priority queues

A tree-of-winners priority queue tolerates insertion of any low or high key value at any time. In contrast, a traditional tree-of-losers priority queue depends on ever-increasing key values. Section 4 introduces an efficient leaf-to-root pass that overcomes this limitation.

## 2.2 Offset-value coding

Offset-value coding [Co77] encodes one row’s key value relative to another key that is earlier in the sort sequence. Offset-value codes are a by-product of comparisons, specifically

in the loser of a comparison. The offset within a loser’s new offset-value code is the position where the keys first differ, e.g., a column index, and the value is the loser’s data value at that offset. Alternatively, the offset is the size of the shared prefix. For example, a duplicate key shares the entire key and thus has an offset equal to the key size.

|  | Column index |   |   |   | Descending<br>offset-value codes |                   |     | Ascending<br>offset-value codes |       |     |
|--|--------------|---|---|---|----------------------------------|-------------------|-----|---------------------------------|-------|-----|
|  | 0            | 1 | 2 | 3 | offset                           | domain<br>- value | OVC | arity -<br>offset               | value | OVC |
| Rows<br>and<br>their<br>column<br>values | 5            | 4 | 7 | 3 | 0                                | 95                | 95  | 4                               | 5     | 405 |
|  | 5            | 4 | 7 | 6 | 3                                | 94                | 394 | 1                               | 6     | 106 |
|  | 5            | 4 | 8 | 5 | 2                                | 92                | 292 | 2                               | 8     | 208 |
|  | 5            | 4 | 9 | 1 | 2                                | 91                | 291 | 2                               | 9     | 209 |
|  | 5            | 4 | 9 | 1 | 4                                | 100               | 500 | 0                               | -     | 0   |
|  | 5            | 5 | 2 | 3 | 1                                | 95                | 195 | 3                               | 5     | 305 |
|  | 5            | 5 | 6 | 7 | 2                                | 94                | 294 | 2                               | 6     | 206 |

Fig. 2: Offset-value codes in a sorted file or stream

Figure 2 illustrates descending and ascending offset-value codes in a stream of rows in ascending sort order on all columns. With four sort columns, the arity of the sort key is 4; the domain of each column is 1 to 99. For an ascending sort order, descending offset-value codes take the actual offset but the negative of the column value, whereas ascending offset-value codes take the negative offset but the actual column value. The first row has offset 0 by definition. Figure 2 ignores that small key domains permit encoding multiple key columns together. IBM’s CFC “compare and form codeword” instruction [IB88, Iy05] supports offset-value coding for a descending sort order of normalized keys (order-preserving byte strings), blocks of bytes as values, and block counts as offsets.

With offsets and values combined as shown in Figure 2, a single integer instruction may decide a comparison. If two rows and their key values *A* and *B* are encoded relative to the same key *C* that is earlier in the sort sequence, and if the offsets of *A* and *B* differ, then the one with the higher offset is earlier in the sort sequence. Otherwise, if the two data values at the common offset differ, then these data values decide the comparison. Otherwise, additional data values in *A* and *B* must be compared.

In a tree-of-losers priority queue with offset-value coding, each tree node lost to the local winner and its local offset-value code is set relative to the local winner. Conversely, the local key value in a tree node other than a leaf was a winner in all nodes up from the leaf where it entered the tree, and all key values along that path are encoded relative to this local key value. The overall winner in the tree’s root is no exception: along its entire leaf-to-root path, all key values lost to (and are encoded relative to) the overall winner.

Merging sorted runs repeatedly replaces the overall winner with its successor from the same merge input. With merge inputs’ fixed assignment to leaf nodes in a tree-of-losers priority queue, a successor retraces the leaf-to-root path of the prior overall winner. As

this successor and all keys on its leaf-to-root path are encoded relative to the prior overall winner, offset-value coding applies to all comparisons in a tree-of-losers priority queue.

Offset-value codes decide many comparisons in a tree-of-losers priority queue. Column value comparisons are required only if two rows have equal offset-value codes. They start after the offset and value encoded in these offset-value codes. After such a row comparison is decided, the loser's offset is incremented by the count of column value comparisons. With  $K$  sort columns, the sum of all offset increments is limited to  $K$  in each row; in an input with  $N$  rows, the sum of all increments and thus the count of all column value comparisons are limited to  $N \times K$ . Importantly, there is no  $\log(N)$  multiplier here. Thus, tree-of-losers priority queues and offset-value coding guarantee that the effort for column value comparisons is linear in the count of rows and in the count of sort columns, quite like the effort for computing hash values in hash-based query execution.

This limit on column value comparisons resonates with data-specific analysis of string sorting [Se10]: the total count of “=” comparisons of symbols (within strings) or of column values (within database rows) equals the opportunity for compression in a sorted dataset. Compression here may be prefix truncation in row storage (e.g., using offset-value coding as shown in Figure 2), run-length encoding of leading columns in column storage, or shared prefixes in a trie [Se10]. Conversions between these alternative formats do not require additional comparisons of symbols or column values [DG22].

Comparisons of offset-value codes are free if they are subsumed in other algorithm activities. In quicksort, for example, the inner-most loop not only compares key values but also looping indexes: when the loops from the left and the right meet, the partitioning step is complete. In priority queues, the inner-most loop compares key values only after testing whether there even are valid key values. This is needed because during queue construction, some entries have not yet been filled; after the end of some merge inputs, some queue entries no longer have valid keys; and during run generation by merging single-row runs, there is only queue build-up and tear-down.

During run generation by continuous replacement selection, the run identifier can prefix the user-defined key as an artificial leading key column. If so, early and late fence values may be modeled as initial and final runs with multiple early and late fence key values, e.g., one for each merge input [Gr06]. All of this can be folded into each row's offset-value code: if two rows have equal offset-value codes, they are both valid (neither early nor late fences), they go to the same output run, they differ from their shared base row (an earlier winner) at the same offset, they have the same value at that offset, and the next step must compare further columns. Thus, comparisons of offset-value codes are not overhead as they simply take the place of testing for fence keys during continuous replacement selection.

The design also reduces CPU cache faults. If a tree-of-losers priority queue requires 8 bytes per entry, an L1 cache can retain a priority queue (an array) of 512 or 1,024 entries. The data records may or may not fit in a lower-level cache but offset-value codes decide many

comparisons without cache fault, many more than traditional pairs of key prefix and data pointer [Hu63, Ny95]. Mini-runs of this size remain in DRAM until merged (with fan-in 512 or 1,024) to form initial runs on temporary external storage [BL89, Fr56, Ny95].

### 2.3 External merge sort

Priority queues enable efficient external merge sort in multiple ways. The most obvious one is merging sorted runs. In fact, merging runs is a perfect application for traditional tree-of-losers priority queues with only complete leaf-to-root passes. A related application is forecasting [Fr56], i.e., predicting which merge input benefits most from an additional input buffer for asynchronous read-ahead. Forecasting tracks, for each merge input, the highest key value read so far and selects the lowest of these values. A third application of priority queues in external merge sort chooses which runs to merge next, e.g., the smallest existing runs [Hä77b] or the set with the most similar sizes. The former heuristic is widely used; the latter heuristic applies when the final input size is not yet known, i.e., when merging while still consuming unsorted input rows.

Another well-known application of tree-of-losers priority queues in external merge sort is run generation, whether in read-sort-write cycles or in continuous replacement selection [Go63]. In read-sort-write cycles, priority queues suffer from repeated build-up and tear-down such that quicksort is often preferred. In cache-optimized read-sort-write cycles, however, merging cache-sized runs in memory to form the initial run on external temporary storage [BL89, Fr56, Ny95] uses a priority queue; moreover, creating many cache-sized runs permits using a tree-of-losers priority queues very efficiently in a way similar to replacement selection. In traditional replacement selection using a single priority queue for all unsorted rows in memory, fixed-size rows enable runs twice the size of memory, but variable-size rows require the techniques introduced in Sections 4 and 6.

Finally, priority queues are useful in many scheduling decisions around external merge sort, e.g., which query to run next, where to grant more memory, where to pinch memory, etc.

## 3 Addressable priority queues

In a scheduling application or a simulation, if a future event is cancelled, an entry in the priority queue must be found and deleted, which requires an addressable priority queue. In a tree-of-losers priority queue, a deletion replaces a valid key value with a late fence.

In a tree-of-losers priority queue, the index is the handle by which to find and identify an entry. As each index maps to a specific leaf node, a search along one leaf-to-root path suffices. The search ends at the sought index value, with a 50% probability that the sought entry was a loser left behind in the leaf, a 25% probability for the parent node, etc. On

average, just less than two nodes are inspected along a leaf-to-root path, for a constant expected time independent of the size or capacity of the priority queue.

If the new key value associated with an index is higher than the one in the priority queue, which is always true if the new key value is a late fence, the leaf-to-root pass can repair the tree-of-losers priority queue and its invariants. In fact, the required logic treats the sub-tree rooted at the sought index as the entire priority queue. The principal code change merely adds a termination condition to the core logic of tree-of-losers priority queues.

```

10. void PQ::pass (Index const index, Key const key)
11. {
12.     Node candidate (index, key);
13.     Index slot;
14.     for (leaf (index, slot); parent (slot),
15.         slot != root () && heap [slot].index != index; )
16.         if (heap [slot].less (candidate))
17.             heap [slot].swap (candidate);
18.     heap [slot] = candidate;
19. }
```

Fig. 3: The modified leaf-to-root pass

Figure 3 highlights the code changes relative to Figure 1, including saving the final candidate in the position of the replaced entry (line 18), not necessarily in the tree's root (line 8 in Figure 1). Note that swaps may occur along the path from leaf to replaced value (lines 16 and 17), that the new loop continuation condition (line 15) could replace rather than augment the original condition, and that line 18 could replace line 8 in Figure 1.

If the new key value associated with an index is lower than the one in the priority queue, the final position of the new key value is not between leaf and replaced entry but between replaced entry and root. Section 4 introduces the required new logic.

## 4 Non-monotone priority queues

If, in an ascending sort, a new key value is higher than the key value it replaces, the input key value cannot go further on its leaf-to-root pass than the position of the replaced key value. If, however, a new key value is lower than the key value that it replaces, including a valid key value replacing a late fence, its final position is on the path between the position of the replaced key value and the root. This requires an initial leaf-to-root pass that ends at the key value to be replaced. This first part equals the search discussed in Section 3, except that the lower key does not swap places with tree entries between the leaf and the replaced value. In fact, the absence of such swaps indicates that the new key value may be smaller than the key value that is about to be replaced in the priority queue.

The new key value may even be lower than some of the key values between the replaced value and the tree's root. If so, such a value might need to move backward on its leaf-to-root

path in order to make room for the new, lower key value. Importantly, only one key value between replaced tree entry and root is a candidate for moving backward, namely the entry that formerly emerged as winner from the sub-tree rooted at the replaced key value. Thus, the first step is to locate this former winner between replaced value and root. The second step compares the new key value with this former winner; if the former winner wins again, then the new value simply overwrites the value to be replaced. Otherwise, the former winner moves backward, overwriting the replaced key value, and the steps repeat.

```

20. void PQ::pass (Index const index, Key const key)
21. {
22.     Node candidate (index, key);
23.     Index slot;
24.     Level level;
25.     for (leaf (index, slot, level); parent (slot, level),
26.          slot != root () && heap [slot].index != index; )
27.         if (heap [slot].less (candidate))
28.             heap [slot].swap (candidate);
29.
30.     Index dest = slot;
31.     if (candidate.index == index)
32.         while (slot != root ())
33.             {
34.                 Level const dest_level = level;
35.                 do { parent (slot, level); }
36.                 while ( ! heap [slot].sibling (candidate, dest_level));
37.
38.                 if (heap [slot].less (candidate)) break;
39.
40.                 heap [dest] = heap [slot];
41.                 dest = slot;
42.             }
43.     heap [dest] = candidate;
44. }
```

Fig. 4: The repair loop added

Figure 4 shows the repair loop (lines 30 to 43) that moves former winners backward on their leaf-to-root path. Instead of saving the final candidate directly (line 18 in Figure 3), the replaced key value becomes the initial destination for the candidate (line 30) and the candidate moves into the heap eventually (line 43). Line 31 ensures that the repair loop runs only if the new key value is small, i.e., after a search loop (lines 25 to 28) without a swap. The repair loop ends when it reaches the tree root (line 32) or when it finds a former winner with a lower key value (line 38). A nested loop searches for a former winner to move backward (lines 35 and 36). After a backward move (line 40), its source becomes the candidate’s new destination (line 41). The tree level is tracked to test whether an ancestor was a former winner at the current destination and therefore could move backward to the current destination (lines 24, 25, and 34 to 36). Extended “leaf” and “parent” methods (lines 25 and 35) initialize the level to 0 and increment it by 1. Even with three loops in total (lines 25, 32, and 35), the complexity of the entire process is still strictly logarithmic like

the height of the tree, with at most  $n$  invocations of the “less” comparison method (lines 27 and 38) in a tree-of-losers priority queue with capacity  $2^n$ .

The conditions for the repair loop permit a few optional optimizations. First, the repair loop is not required if the old key value is an early fence or if the new key value is a late fence. Second, the four conditions for entering the loop could be reordered by probability and execution cost [Ha77a]. Third, the loop’s continuation test could move to the bottom.

```

45.   Index dest = slot;
46.   if (slot != root () && candidate.index == index &&
47.       key != late_fence (index) && heap [slot].key != early_fence (index))
48.       do
49.       {
50.         ...
51.       } while (slot != root ());
52.   heap [dest] = candidate;

```

Fig. 5: Alternative conditions for the repair loop

Figure 5 shows the relevant code fragments with these optional optimizations in lines 46 and 47. Lines 45 and 52 are lines 30 and 43 in Figure 4, respectively. Line 50 stands for lines 34 to 41 in Figure 4.

## 5 Double-ended priority queues

The example problem to be solved here is the following: a number of sellers frequently change their asking prices (for some goods or service); the lowest-price sellers often sell out and must withdraw their offer or raise their asking prices; and the highest-price sellers often withdraw their uncompetitive offers or drop their asking prices. Both buyers and sellers want the current lowest and highest asking prices readily available.

The solution employs an array of sellers with their current asking price if any. In addition, two priority queues, one ascending and one descending, track the lowest and highest asking prices. Together, they form a double-ended priority queue.

Both priority queues use the same indexes as the array, must be addressable to support deletion by index, and must be non-monotone to track all possible changes in asking prices, both increases and decreases.

Seeing current lowest and highest asking prices requires “top” methods. The lowest-price seller withdrawing requires a “pop” method in the ascending priority queue and a “delete” method in the descending priority queue. The highest-price seller withdrawing is just the opposite. Any other seller withdrawing requires a “delete” method in both priority queues. Any change in asking price requires an “update” method in both priority queues.

All of these methods invoke the “pass” method of Figure 4. The exception is “pop”: it might just replace a valid key value in the tree root with an early fence, but then subsequent “pop”

or “top” invocations must invoke the “pass” method with the appropriate index and a late fence to propel a valid key value to the tree root. “Delete” also invokes “pass” with a late fence and “update” invokes “pass” with the new asking price.

Each invocation of the “pass” method requires time at most linear with the height and logarithmic with the capacity of the priority queue. The “delete” and “update” methods inspect just less than two tree nodes on average for constant expected time (independent of the size of the priority queue). The effort required in “delete” is likely less when invoked to match a “pop” in the other priority queue, but deleting the entry at a tree root forces a complete leaf-to-root pass in one of the priority queues. An optimization avoids or delays this full pass by placing an early fence in the root node.

The following experiments simulate up to 1,024 sellers (tree height 2-10) and  $2^{25} \approx 33.5M$  changes in asking prices. They ran the code of Figures 4 and 5 on an Intel Celeron N3060 CPU (launched in 2016, 1.6 Ghz base frequency, 2.48 GHz burst frequency).

| Tree height | Low & high | Random |
|-------------|------------|--------|
| 2           | 1.250      | 1.094  |
| 3           | 1.438      | 1.266  |
| 4           | 1.687      | 1.359  |
| 5           | 1.906      | 1.484  |
| 6           | 2.234      | 1.484  |
| 7           | 2.390      | 1.516  |
| 8           | 2.640      | 1.531  |
| 9           | 2.843      | 1.531  |
| 10          | 3.046      | 1.500  |

Fig. 6: Maintenance effort for changing key value [CPU seconds]

Figure 6 shows the CPU times in seconds for two experiments. The first experiment (middle column) assumes that only lowest- and highest-price sellers change their asking prices. In other words, the double-ended priority queue runs “pop” and “delete” followed by an “insert” in both priority queues. For tree capacities of 4 and 1,024, i.e., tree heights of 2 and 10, the time difference is 1.796 seconds (3.046–1.250). Multiplying with the burst frequency (2.48 GHz) and dividing by the count of changes in asking prices ( $2^{25}$ ) as well as the difference in tree heights (10–2) suggests that each tree level adds only about 17 CPU cycles to the effort of maintaining a double-ended priority queue constructed from two tree-of-losers priority queues.

The second experiment (right column) assumes that all sellers randomly change their asking prices. Recall that a randomly chosen seller (index in a priority queue) has remained in a tree leaf with a 50% probability, in a leaf’s parent with a 25% probability, etc., for an average search depth of just under two tree nodes. This is independent of the tree height, and indeed the elapsed times remain fairly steady. The differences observed are more likely a result of array sizes and CPU caches than of algorithm or code complexity.

## 6 Offset-value coding for non-monotone priority queues

Offset-value coding speeds up sorting with tree-of-losers priority queues. In fact, Section 2.2 shows how, in an external merge sort for  $N$  rows with  $K$  key columns, these techniques reduce worst-case counts of column value comparisons from  $O(K \times N \log N)$  to  $N \times K$ . It has been unclear, however, whether scheduling and sorting applications with complex keys can benefit similarly from offset-value coding. Of course, no crisp benchmark and complexity metric exist for scheduling, but can offset-value coding reduce the comparison effort and can comparisons skip over column values already compared earlier?

The main difference to merging sorted runs is that predecessors and successors in a merge input are sorted and their offset-value codes can be known. In contrast, in scheduling applications, predecessor key values may not be known and successors may not have offset-value codes. In fact, as offset-value coding always is relative to a smaller key, a replacement value smaller than its predecessor cannot possibly have an offset-value code. A replacement key value without offset-value code relative to its predecessor (for the same index) requires full comparisons, i.e., starting at offset 0 within the key.

Fortunately, such a replacement key value still requires only a single leaf-to-root pass with a search loop (lines 25 to 28 in Figure 4) and a repair loop (lines 30 to 43 in Figure 4). Just as fortunately, full comparisons without the benefit of offset-value coding are required only until the search loop swaps a replacement key value into its correct location within the tree-of-losers structure. Thereafter, all comparisons benefit from offset-value coding. A replacement key value belongs into the leaf with 50% probability, into the leaf's parent with 25% probability, etc., for only about two full comparisons on average.

The first swap puts the new key value into its correct place within the tree-of-losers priority queue. After a swap, no repair loop is needed (see line 31 in Figure 4 and line 46 in Figure 5). A repair loop, if needed, may move some key values backward on their leaf-to-root paths (see Section 4 and Figure 4). In this case, offset-value codes along the leaf-to-root path must be adjusted such that losers are always encoded relative to the correct winner.

Figure 4 shows the traditional search loop (lines 25 to 28) and the new repair loop (lines 30 to 43). The traditional search can maintain offset-value codes in the traditional way: if column value comparisons are required, the loser's offset increases by their count. The comparison logic (line 27) can readily adjust the loser's offset-value code in this way.

Maintenance of offset-value codes in the repair loop seems expensive when a former winner moves backward on its leaf-to-root path, skipping backward over other tree nodes and key values (the ones skipped in lines 35 and 36 of Figure 4). When a new key value emerges as the new winner, existing skipped-over offset-value codes must be re-encoded relative to the new winner, incurring full row comparisons and thus column value comparisons.

Fortuitously, a recent theorem [GD22] on offset-value codes supplies a remedy: for three sorted key values  $A < B < C$ , the offset-value code of the third key value rela-

tive to the first one is the extreme of the other two offset-value codes, or  $ovc(A, C) = \max(ovc(A, B), ovc(B, C))$  for ascending offset-value codes<sup>4</sup>. When a former winner moves backward, the new winner is lower (earlier in the sort order) than the former winner, which in turn is lower than the skipped-over key values between the former winner's old and new locations along the leaf-to-root path. If the new winner is key value  $A$  in the theorem, the former winner is key value  $B$ , and each skipped-over key value is key value  $C$ , then the new offset-value codes for the skipped-over key values is simply the maximum of their existing offset-value codes relative to the former winner and the offset-value code of the former winner relative to new key value.

In order to adjust those offset-value codes, the repair loop of Figure 4 needs another nested loop. Its range matches the first nested loop (lines 35 and 36 in Figure 4). The new nested loop does not change the complexity of the process even if it is no longer strictly true that a tree-of-losers priority queue can absorb any new key in a single leaf-to-root pass.

```

53. inline void setMax (Key & x, Key const y) { if (x < y) x = y; }
54.
55. void PQ::pass (Index const index, Key const key, bool full_comp)
56. {
57.     Node candidate (index, key);
58.     Index slot;
59.     Level level;
60.     for (leaf (index, slot, level); parent (slot, level),
61.          slot != root () && heap [slot].index != index; )
62.         if (heap [slot].less (candidate, full_comp))
63.             heap [slot].swap (candidate), full_comp = false;
64.
65.     Index dest = slot;
66.     if (candidate.index == index)
67.         while (slot != root ())
68.             {
69.                 Level const dest_level = level;
70.                 do { parent (slot, level); }
71.                 while ( ! heap [slot].sibling (candidate, dest_level));
72.
73.                 if (heap [slot].less (candidate, full_comp)) break;
74.
75.                 heap [dest] = heap [slot];
76.                 while (parent (dest), dest != slot)
77.                     setMax (heap [dest].key, heap [slot].key);
78.             }
79.     heap [dest] = candidate;
80. }
```

Fig. 7: Repair of offset-value codes added

<sup>4</sup> This theorem implies Iyer's "unequal value theorem" [Iy05]: After offset-value codes decide a row comparison, the loser retains its offset-value code. Formally:  $ovc(A, B) < ovc(A, C) \Rightarrow ovc(B, C) = ovc(A, C)$ . Proof:  $ovc(A, C) = \max(ovc(A, B), ovc(B, C)) \wedge ovc(A, C) \neq ovc(A, B) \Rightarrow ovc(A, C) = ovc(B, C)$ .

Figure 7 adds a parameter to the “pass” method (line 55) to indicate whether full comparisons ought to be used in the “less” method (lines 62 and 73). If set initially, it remains true until the new key value is in its correct place after the first swap (line 63).

Figure 7 also adds the new loop (lines 76 and 77). The variable “dest” is abused as a looping variable, yet the loop’s continuation condition (“dest != slot” in line 76) echoes the direct move that the loop replaces (“dest = slot” in line 41 of Figure 4). The “parent” method in line 76 is the same as in Figures 1 and 3. The loop body (line 77) applies the theorem to any skipped-over offset-value codes. The “key” field in each tree entry is the offset-value code relative to the local winner. Auxiliary method “setMax” is given in line 53.

Like the key comparison in the search loop (line 62), the “less” method in the repair loop (line 73) must set the loser’s new offset-value code if column value comparisons are required. If the candidate wins, then the key value in the heap is the loser and must be encoded relative to the candidate when it travels backward to the current destination (line 75). Otherwise, the candidate goes back to the current destination (line 79) and must be encoded relative to the (old and new) winner.

In a general scheduling application and its priority queues, many index values will be active and inactive at various times. In a merge step, e.g., in an external merge sort, some of the merge inputs might disconnect from the merge and reconnect later, e.g., during a key range that is not represented in one (or several) of the input runs. A tree-of-losers priority queue models such inactive index values using invalid keys or fences with effective key value  $+\infty$ . But how does offset-value coding deal with fence keys? What are the offset-value codes for a valid key value relative to an early fence and for a late fence relative to a valid key value?

The solution follows directly from the role of fences: artificial keys preceding the first row and succeeding the last row in a sorted run. If all column values in fences are  $-\infty$  or  $+\infty$ , then the first difference with any valid key value is at offset 0 and the value at that offset is  $+\infty$  in a late fence. With these definitions, all algorithms above, including calculation of offset-value codes, work not only with valid key values but also with fences.

There is one important difference for “pop” and “delete” operations, however. Without offset-value coding, deletion of a valid key value in the root node can simply replace the key value with an early fence. With offset-value coding, offset-value codes in its leaf-to-root path require repair. This new leaf-to-root pass resets all offsets to 0 (because the winner is or would have been the new early fence in the root node); the value is the first column or  $\pm\infty$  for fence keys. Alternatively, instead of a “pop” operation, a “top” operation is more desirable and is sufficient if the next operation is a “push” for the same index. For “delete” operations, the optimization above must be disabled in a priority queue with offset-value coding such that deletion always invokes a leaf-to-root pass, even when deleting the key value in the tree’s root.

## 7 Replacement selection for variable-size records

In the context of external merge sort and specifically run generation, multiple situations require the repair loop of Figure 4. First, in run generation for fixed-size records arriving and evicted in groups (e.g., pages) rather than one record at a time, the logic of Figure 3 can replace multiple entries in the tree by late fences until a group is complete. When new valid key values replace these late fences, the repair logic of Figure 4 is required because all valid key values sort lower than late fences. If the external merge sort uses offset-value coding to reduce column value comparisons to  $N \times K$ , it requires the new logic of Figure 7.

Second, run generation with variable-size records may require evicting multiple records from the in-memory workspace before it can absorb an arriving large record. In the opposite situation, when the sort logic evicts a large record from the workspace, multiple arriving small records might be inserted, replacing multiple late fences with valid key values.

Third, if a key range of non-trivial size occurs in only one of the merge inputs, there is no need to “merge” each key in this range. After observing successive overall winners from the same merge input, one might want to know the runner-up key and then scan or skip forward within the winning input. One way to determine the runner-up pushes the runner-up to the root by using a late fence to fake the winner’s end-of-input. Re-introducing the former winner into the tree-of-losers priority queue after moving a key range directly to the merge output requires the new logic of Figure 7.

| Group size | Row comparisons |         | OVC comparisons |         |
|------------|-----------------|---------|-----------------|---------|
|            | Total [M]       | Per row | Total [M]       | Per row |
| 1          | 335.56          | 10.00   | 335.55          | 10.00   |
| 2          | 357.53          | 10.66   | 318.78          | 9.50    |
| 4          | 380.58          | 11.34   | 306.96          | 9.15    |
| 8          | 398.25          | 11.87   | 300.25          | 8.95    |
| 16         | 409.68          | 12.21   | 296.79          | 8.85    |
| 32         | 416.68          | 12.42   | 295.26          | 8.80    |
| 64         | 420.85          | 12.54   | 294.57          | 8.78    |
| 128        | 423.43          | 12.62   | 295.49          | 8.81    |
| 256        | 424.06          | 12.64   | 295.11          | 8.80    |

Fig. 8: Priority queue comparison counts with various group sizes

Figure 8 shows counts of comparisons in a priority queue during run generation in an external merge sort. There are  $2^{25} \approx 33.5M$  input rows; the priority queue holds  $2^{10} = 1,024$  entries. The run generation logic deletes groups of rows (to fill an output page) and re-inserts the same count of rows (from an input page). The group or page size varies from 1 row, i.e., traditional replacement selection, to 256 rows. The center pair of columns in Figure 8 clearly shows that grouped removal and insertion into priority queues is somewhat less efficient than traditional replacement selection. Large group or page sizes require about 25% more comparisons.

The right pair of columns in Figure 8 shows how many comparisons benefit from offset-value coding and the new techniques of Section 6 and Figure 7. Their practical impact depends on column count, each column's count and distribution of distinct values, etc. This experiment aims to assess the new techniques independently of these parameters. It is obvious that in one-for-one replacement, practically all comparisons benefit from offset-value coding. For larger group or page sizes, the benefit remains substantial as about 70% of all comparisons might be decided by offset-value codes alone, meaning that in practical settings the new techniques saves substantially more than half of the effort for row comparisons.

## 8 Summary and conclusions

In summary, prior work has shown that

1. a tree-of-losers priority queue requires fewer comparisons than an equivalent traditional tree-of-winners priority queue;
2. internal and external merge sort with tree-of-losers priority queues comes very close to the provable lower bound of  $\log_2(N!)$  row comparisons for  $N$  rows;
3. offset-value coding reduces column or byte comparisons in large keys;
4. the count of column value comparisons in internal and external sorting with offset-value coding is bounded by  $N \times K$  for sorting  $N$  rows with  $K$  key columns, i.e., linear in the count of rows and the count of key columns; and
5. the core logic for a traditional leaf-to-root pass in a tree-of-losers priority queue is small, simple, and efficient.

In addition, new work shows how

6. data-specific analysis of string sorting [Se10] applies equally to prefix sharing in a trie of strings, to offset-value coding in a sorted database table, and to the count of symbol comparisons or of column value comparisons in an internal or external merge sort using tree-of-losers priority queues and offset-value coding;
7. a small extension turns tree-of-losers priority queues into addressable priority queues, i.e., any key value can increase at any time and can be logically deleted at any time;
8. a larger extension turns tree-of-losers priority queues into non-monotone priority queues, i.e., new key values may be lower or higher than earlier key values;
9. a third extension enables efficient incremental maintenance of offset-value codes, even in cases of non-monotone sequences of input key values;
10. the first two extensions permit very efficient double-ended priority queues, among many other scheduling applications within and beyond database systems; and

11. all three extensions together enable new flexibility and efficiency in external merge sort for query processing, index creation and maintenance, database reorganization, and data processing frameworks such as MapReduce and its many successors.

In conclusion, the new techniques for tree-of-losers priority queues and offset-value coding complement recent innovations in database query processing. These innovations include passing offset-value codes from one query execution operation to the next, thus reducing most of their matching logic to comparisons of compiled-in integers rather than comparisons of large records with complex fields and expensive comparison logic [DG22, GD22]. For example, when a query like “. . . count (distinct. . .). . . group by. . .” requires first duplicate removal and then grouping, these operations can share not only the sort but also offset-value codes such that the grouping logic never compares column values. For a second example, merge joins of sorted scans or streams can avoid many column value comparisons, perform their required row comparisons using offset-value codes just as efficiently as a hash join does using hash values, and save lots of CPU effort, memory, and overflow (compared to a hash join). In a variation of this example, complex-object assembly by multiple merge joins on the same column can use offset-value codes in all join operations if the join logic produces offset-value codes for its sorted output even if a join suppresses some input rows and duplicates others [GD22]. For a third example, if grouping on a foreign key can be “pushed down” to a join input, early aggregation and wide merging [DGN22] can speed up the sort and offset-value codes from the sort can speed up the merge join. On the other hand, if a grouping operation cannot be pushed down but the merge join produces offset-value codes for its output, then grouping on the join column(s) never needs to compare column values. More examples readily come to mind.

Put differently, wherever query planning can exploit interesting orderings in storage structures and intermediate query results [Se79], query execution can exploit offset-value codes [Co77]. Just as any industrial-grade state-of-the-art query optimizer exploits interesting orderings, query execution should exploit offset-value codes. Although both concepts originated in the 1970s, almost half a century ago, they were linked only recently by widening the scope of offset-value coding from merge sort to all sort-based query execution algorithms. Together, recent innovations for priority queues and for offset-value coding enable sort-based query processing to compete with hash-based query processing and to shine even for large unsorted inputs with large keys.

## 9 Acknowledgements

Jim Gray suggested exploring offset-value coding. Wey Guy and Thanh Do requested some of the explanations in this paper. Raimund Seidel pointed out bottom-up heapsort and data-specific algorithm analysis. Their help and contributions are gratefully acknowledged.

## References

- [BE77] Blasgen, Mike W.; Eswaran, Kapali P.: Storage and access in relational data bases. *IBM Syst. J.*, 16(4):362–377, 1977.
- [BL89] Baer, Jean-Loup; Lin, Yi-Bing: Improving Quicksort performance with a codeword data structure. *IEEE Trans. Software Eng.*, 15(5):622–631, 1989.
- [Br84] Bratbergsgengen, Kjell: Hashing methods and relational algebra operations. In: *VLDB*. pp. 323–333, 1984.
- [Co77] Conner, W. M.: Offset-value coding. In: *IBM Technical Disclosure Bull.* pp. 2832–37, 1977.
- [De84] DeWitt, David J.; Katz, Randy H.; Olken, Frank; Shapiro, Leonard D.; Stonebraker, Michael; Wood, David A.: Implementation techniques for main memory database systems. In: *ACM SIGMOD*. pp. 1–8, 1984.
- [DG85] DeWitt, David J.; Gerber, Robert H.: Multiprocessor hash-based join algorithms. In: *VLDB*. pp. 151–164, 1985.
- [DG22] Do, Thanh; Graefe, Goetz: Robust and efficient sorting with offset-value coding. Accepted for publication in *ACM TODS*, March 2022.
- [DGN22] Do, Thanh; Graefe, Goetz; Naughton, Jeff: Efficient sorting, duplicate removal, grouping, and aggregation. *ACM TODS*, 47(4), December 2022.
- [Ep79] Epstein, Robert: Techniques for processing of aggregates in relational database systems. In: *Univ. of California at Berkeley, UCB/ERL Memorandum M79/8*. 1979.
- [Fr56] Friend, Edward H.: Sorting on electronic computer systems. *J. ACM*, 3(3):134–168, 1956.
- [GD22] Graefe, Goetz; Do, Thanh: Offset-value coding in database query processing. submitted for publication, September 2022.
- [Go63] Goetz, Martin A.: Internal and tape sorting using the replacement-selection technique. *Commun. ACM*, 6(5):201–206, 1963.
- [Gr06] Graefe, Goetz: Implementing sorting in database systems. *ACM Comput. Surv.*, 38(3), 2006.
- [Ha77a] Hanani, Michael Z.: An optimal evaluation of Boolean expressions in an online query system. *Commun. ACM*, 20(5):344–347, 1977.
- [Hä77b] Härder, Theo: A scan-driven sort facility for a relational database system. In: *VLDB*. pp. 236–244, 1977.
- [Ho62] Hoare, C. A. R.: Quicksort. *Comput. J.*, 5(1):10–15, 1962.
- [Hu63] Hubbard, George U.: Some characteristics of sorting computing systems using random access storage devices. *Commun. ACM*, 6(5):248–255, 1963.
- [IB88] IBM: Enterprise system architecture/370, principles of operation. IBM publication SA22-7200-0, 1988.

- [IS56] Isaac, Earl J.; Singleton, Richard C.: Sorting by address calculation. *J. ACM*, 3(3):169–174, 1956.
- [Iy05] Iyer, Bala R.: Hardware assisted sorting in IBM’s DB2 DBMS. In: *International Conference on Management of Data (COMAD)*. 2005.
- [Kn98] Knuth, Donald Ervin: *The art of computer programming, Volume III: sorting and searching*, 2nd edition. Addison-Wesley, 1998.
- [KTM83] Kitsuregawa, Masaru; Tanaka, Hidehiko; Moto-Oka, Tohru: Application of hash to data base machine and its architecture. *New Gener. Comput.*, 1(1):63–74, 1983.
- [LG98] Larson, Per-Ake; Graefe, Goetz: Memory management during run generation in external sorting. In: *ACM SIGMOD*. pp. 472–483, 1998.
- [NKT88] Nakayama, Masaya; Kitsuregawa, Masaru; Takagi, Mikio: Hash-partitioned join method using dynamic destaging strategy. In: *VLDB*. pp. 468–478, 1988.
- [Ny95] Nyberg, Chris; Barclay, Tom; Cvetanovic, Zarka; Gray, Jim; Lomet, David B.: AlphaSort: a cache-sensitive parallel external sort. *VLDB J.*, 4(4):603–627, 1995.
- [Se79] Selinger, P. Griffiths; Astrahan, M. M.; Chamberlin, D. D.; Lorie, R. A.; Price, T. G.: Access path selection in a relational database management system. In: *ACM SIGMOD*. p. 23–34, 1979.
- [Se10] Seidel, Raimund: Data-specific analysis of string sorting. In: *ACM-SIAM SODA*. pp. 1278–1286, 2010.

# Workload-Driven Data Placement for Tierless In-Memory Database Systems

Ben Hurdelhey<sup>1</sup>, Marcel Weisgut<sup>2</sup>, Martin Boissier<sup>3</sup>

**Abstract:** High main memory consumption is a significant cost factor for in-memory database systems. Tiering, i.e., placing parts of the data on memory or storage devices other than DRAM, reduces the main memory footprint. A controlled data placement can assign rarely accessed data to slow devices while frequently used data remains on fast devices, such as main memory, to maintain acceptable query latencies. We present an automatic data placement decision system for the in-memory database Hyrise. The system organizes the memory and storage devices in a tierless pool, with no fixed device class categorization or performance order. The system supports data placement use cases, such as minimizing end-to-end query latencies and making cost-optimal purchase recommendations in cloud environments. In this paper, we introduce an efficient calibration process to derive cost models for various storage devices. To determine data placements, we introduce a linear programming-based approach, which yields optimal configurations, and an efficient heuristic. With a set of main memory and SSD devices, we can reduce the main memory consumption for base table data of the TPC-DS benchmark by 74 percent when accepting a workload latency increase of 52 percent. In a comparison of data placement algorithms and cost models, we find that simplistic algorithms (e.g., greedy algorithms) can present viable alternatives to optimal linear programming algorithms, especially under cost prediction inaccuracies.

**Keywords:** Tiering; Data Placement; In-Memory Database Systems; Linear Programming; Cost Models

## 1 Data Placement for In-Memory Database Systems

In contrast to traditional disk-based database management system (DBMS), in-memory database management systems (IMDBMSs) store their data in dynamic random-access memory (DRAM) instead of comparatively slow hard disk drives (HDDs) or solid state drives (SSDs). Holding all data in main memory allows for faster query processing, which in turn facilitates business applications, for example, by flexibly computing aggregates on-the-fly [ÖTT17, PI14]. However, IMDBMSs inherently come with high main memory consumption, which can result in increased operating costs for pure in-memory database systems [Lo19]. The continuously growing amounts of data aggravate this problem, increasing the need for larger-than-memory DBMS [Ma16]. Furthermore, DRAM capacity growth is slowing down and is expected to reach an upper limit [Ma02, Sh20].

Concluding, we argue that it can be desirable to reduce the main memory consumption of IMDBMS. Tiering, i.e., placing selected data on other memory/storage devices with a lower cost per dollar, is a viable strategy to reduce the DRAM consumption [Du16]. When

<sup>1</sup> Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Deutschland ben.hurdelhey@student.hpi.de

<sup>2</sup> Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Deutschland marcel.weisgut@hpi.de

<sup>3</sup> Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Deutschland martin.boissier@hpi.de

moving data from DRAM to devices such as persistent memory (PMem) or SSD, we have to keep in mind that these memory/storage devices offer worse performance characteristics than DRAM, i.e., higher latencies and lower bandwidths. Moving the entire stored database to another device could therefore cause substantial data access cost increases and, thus, increased query latencies. However, data access in database queries is often highly skewed because some part of the data is frequently accessed while another part is only rarely or never queried. This skew is demonstrated by analyses of production database systems by Höppner et al. [HWR14, p. 68], Dreseler [Dr22, p. 12], and Boissier et al. [BSU18, p. 210]. With tiering, we can exploit data access skew by preferably storing the infrequently accessed data on the slower but less expensive memory/storage devices. The critical challenge is determining a *data placement* [Dr22, Vo20], i.e., an assignment of data to devices, to minimize the runtime performance impacts while reducing the DRAM usage. We refer to the series of instructions that determines a data placement as the *placement algorithm*.

We make the following contributions to advance automatic data placement for IMDBMS.

- We propose a placement system for columnar relational in-memory database systems with horizontally partitioned tables. The system is based on linear programming (LP) algorithms and supports multi-constraint multi-device data placement decisions before and after the hardware purchase (Sect. 2). We exemplarily implemented the placement selection system for the open-source in-memory research database Hyrise [Dr19].
- For efficient and accurate placement cost prediction, we propose and evaluate a calibrated cost model based on access tracking data, which allows for efficient cost prediction (Sect. 3).
- We propose and compare multiple placement algorithms and determine a Pareto-optimal trade-off between resource efficiency and result quality, demonstrating that simple algorithms (e.g., Greedy, Knapsack) can be viable alternatives to optimal linear programming algorithms (Sect. 4).

## 2 Automatic Placement Decisions

Our data placement approach has the following general characteristics.

- The data placement module works **autonomously**. Manual database administration is laborious and error-prone [Ma21]. Placement decisions can be complex, and it can be infeasible to manually determine the optimal data placement.
- While it is possible to place temporary data structures used during query processing on secondary devices (i.e., devices other than DRAM) [Da21], we focus on evicting append-only data structures of the database system's base tables. Eviction of temporary data structures is required for database operator execution when an operator requires more DRAM than available.
- Our goal is to move infrequently accessed data to slower devices to maintain acceptable query latencies. The *workload* of the database, i.e., the queries being run, determines the *access characteristics* of the stored data. We incorporate access tracking information. Thus, our approach is **workload-driven**.

- While some of the related research has considered only two devices [BSU18, Dr22, La22], we examine data placement techniques that support an **arbitrary number of devices**. In addition, we organize the devices in a **tierless pool**, with no fixed categorization or performance ordering of the devices, similar to [Vo20]. Contrarily to the traditional memory and storage hierarchy shown in Fig. 1a, we regard the devices as separate entities with different access characteristics, as illustrated in Fig. 1b. The term *tiering* is widely used and understood [BSU18, Dr22, Du16]. However, we use the term *devices* instead of *tiers* to not imply any fixed categorization or ordering of the devices by their price or access performance<sup>4</sup>.

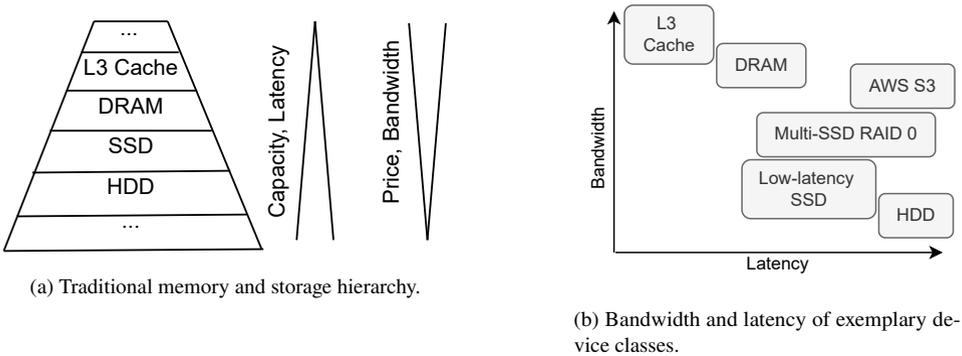


Fig. 1: Tierless device pool concept. Device characteristics based on [Am21, Ap19, Dr22, Wu21] and fio [Ax22] measurements for the devices in Sect. 2.2 as specified at <https://github.com/benrobby/hyrise-data-placement#fio>.

## 2.1 Data Placement in Hyrise

We propose a data placement selection module for Hyrise. The module’s main objective is to support placement algorithm and cost model experiments. The placement module consists of a plugin for Hyrise, which interacts with the database system, and a standalone module to determine the data placements<sup>5</sup>.

Hyrise is a columnar in-memory database system. As described by Dresler et al. [Dr19, p. 316], tables are horizontally partitioned into *chunks*, whereas chunks are mutable as long as data is added and become immutable and read-optimized once their capacity is reached. With the default chunk size used in this work, a single chunk stores 65 535 tuples. Each chunk is vertically partitioned into segments, whereas each segment corresponds to one fraction of a column of the table. Each segment can be encoded independently. Dictionary encoding is the default encoding in Hyrise, which we also use in this work. Hyrise uses

<sup>4</sup> The Encyclopædia Britannica defines a *tier* as “a row or layer of things that is above another row or layer”. The definition can be found at <https://www.britannica.com/dictionary/tier>

<sup>5</sup> More information on the placement selection module for Hyrise can be found at <https://github.com/benrobby/hyrise-data-placement>.

multi-version concurrency control (MVCC) to isolate concurrent transactions. Instead of updating a row, a new row is written, and the old row is marked as invalid [Dr19, p. 320].

The *placement granularity* defines the data unit at which we make placement decisions and move data between tiers. We make placement decisions at a segment granularity to capture both unused columns and vertical skew with multi-dimensional access tracking and placement decisions [Dr22, pp. 97–100].

Access tracking allows observing the access frequencies of stored data. This information is crucial for deciding on which device a specific part of the data is to be stored on. Corresponding to the segment granularity of the placement decisions, we track data accesses for each individual segment using Hyrise’s per-segment access counters [Dr22, pp. 102–104]. For each segment, Hyrise maintains access counters for sequential, monotonic, random, and point *access patterns* [Dr22, p. 92].

Hyrise uses C++17’s polymorphic memory resources (PMRs) to encapsulate the allocation behavior on different devices and add eviction capabilities to arbitrary data structures [Dr22, p. 79-81]. We supply custom memory resources to allocate and deallocate memory on given devices. This approach is based on an existing implementation for Hyrise [We22]. The memory resources use jemalloc to manage the memory allocations and deallocations. In particular, the resource for block devices supplies hooks (i.e., function pointers) to jemalloc to control the underlying memory allocations of the memory allocator. These hooks allocate memory from memory-mapped UMap regions corresponding to one file on the given device (e.g., on an SSD). *UMap* [Pe19] is a user-space page fault handler that allows for configurations such as adjusting the page size or buffer size. Limiting the memory mapped buffer size gives us control over UMap’s eviction behavior. For our experiments, restricting the memory mapped buffer size is crucial so a device does not degenerate to a buffer that can hold all stored data in DRAM. Therefore, we limit the UMap buffer size to 250 MB. Previous research found the optimum page size to be workload-dependent [We22, p. 1205]. However, we use a fixed page size of 128 KiB as this configuration is not our research focus. With the segment granularity, we migrate given segments between devices during the runtime of the DBMS. The database system might perform work during the migration and even access the exact segment currently being migrated. For this reason, we first copy the segment to the new device and then replace the old segment in the chunk with the new segment using an `std::atomic_store` [Dr22, p. 83].

## 2.2 Multi-Objective Data Placement with Linear Programming

In modern cloud environments, traditional on-premise assumptions regarding the device purchase no longer apply. For example, the device hardware is no longer purchased and used throughout the entire device lifespan. Instead, cloud environments allow users to easily migrate between different compute and storage hardware<sup>6</sup> with pay-as-you-go pricing models. This ability to flexibly reconfigure the used devices allows for new data placement

---

<sup>6</sup> For example, virtual machines can be rented and migrated on Amazon AWS (<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-resize.html>).

applications throughout the entire DBMS life-cycle. Our placement selection module supports the following three use cases, which we refer to as *objectives*. For objective O1, we assume the devices are already purchased, and their respective byte capacities constrain the data placement. The goal is to achieve the best possible query runtime performance within the memory budget. Objective O2 occurs during the purchase phase of database deployment. This objective aims to determine purchase recommendations that satisfy a latency constraint for the workload runtime while minimizing the monetary costs for the memory and storage devices. Objective O3 aims to determine purchase recommendations that minimize the predicted query runtimes given a fixed monetary budget for memory/storage devices. We formulate these placement selection problems as linear programming models.

**Objective O1: Byte Capacity Constrained Placement Configurations** The first objective that our solution supports is minimizing the end-to-end runtime of the database system for a given set of devices, each with a fixed byte capacity. The task of the placement selection algorithm is to determine values for the decision variable  $x_{t,a,p,d}$ , which corresponds to assigning a segment in table  $t$ , attribute  $a$ , partition  $p$ , to a device  $d$ .  $T$  and  $D$  are the numbers of tables and devices.  $A$  and  $P$  are the maximum numbers of attributes and partitions over all tables. For these numeric parameters, we use the notation  $t \in T$ , which is equivalent to  $t = 1, \dots, T$ , for improved readability. For example, for three devices,  $d \in D \equiv d \in \{1, 2, 3\}$ .

$$\text{minimize}_{x_{t,a,p,d} \in \{0,1\}^{T \times A \times P \times D}} \sum_{\substack{t \in T, a \in A, \\ p \in P, d \in D}} x_{t,a,p,d} \cdot c_{t,a,p,d} \quad (1)$$

$$\text{s.t.} \quad \sum_{\substack{t \in T, a \in A, \\ p \in P}} x_{t,a,p,d} \cdot s_{t,a,p} \leq b_d \quad \forall d \in D \quad (2)$$

$$\left( \sum_{d \in D} x_{t,a,p,d} \right) = i_{t,a,p} \quad \forall t \in T, a \in A, p \in P \quad (3)$$

The objective (1) of the integer linear programming (ILP) model is to minimize the predicted costs  $c_{t,a,p,d}$  of all segment assignments. Depending on the cost model's accuracy, the objective value can become an accurate runtime prediction for data placements. The LP objective is subject to two constraints. The constraint (2) ensures that the capacity of each device  $b_d$  is not exceeded by the accumulated segment size  $s_{t,a,p}$  of the segments assigned to it. Furthermore, the constraint (3) requires the model to assign each segment to exactly one device:  $i_{t,a,p} \in \{0, 1\}$  is a binary function guaranteeing that only existing segments (due to some tables having more attributes than other tables) will be assigned to a device. Contrarily, non-existing segments will not be assigned to any device.

**Objective O2: Latency Constrained Buying Recommendations** Objective O2 assumes that the user, i.e., the database administrator, wants to pose latency constraints on the database system and spend as little money as possible on devices to satisfy these constraints. In our case, we select the latency constraint that the cumulative runtime of all queries in a

given workload must not exceed a given maximum latency. The algorithm minimizes the dollar costs for the devices required to comply with the given latency constraints. For this, the algorithm determines the hypothetical data placement that satisfies the given latency constraint. In the resulting data placement, the memory or storage usage per device will become the buying recommendation for the memory/storage devices.

$$\text{minimize}_{x_{t,a,p,d} \in \{0,1\}^{T \times A \times P \times D}} \sum_{\substack{t \in T, a \in A, \\ p \in P, d \in D}} x_{t,a,p,d} \cdot s_{t,a,p} \cdot g_d \quad (4)$$

$$\text{s.t.} \quad \sum_{\substack{t \in T, a \in A, \\ p \in P, d \in D}} x_{t,a,p,d} \cdot c_{t,a,p,d} \leq o \quad (5)$$

$$\left( \sum_{d \in D} x_{t,a,p,d} \right) = i_{t,a,p} \quad \forall t \in T, a \in A, p \in P \quad (6)$$

The ILP model is similar to the O1 model. We introduce the maximum runtime cost value  $o$  and the variable  $g_d$  for the cost of a device  $d$  in dollars per byte. The objective (4) of this ILP model is to minimize the total dollar cost of the data placement. The model calculates the dollar cost for a specific data placement using the segment sizes in bytes and the respective device prices in dollars per byte. The constraint (5) calculates the total predicted runtime cost value and poses an upper limit to this cost.

Our cost model's runtime predictions are inaccurate. Therefore, we infer the maximum runtime cost value  $o$  from a given maximum end-to-end latency using experimentally-determined linear interpolation. Calculating the parameters for this formula requires measurements of end-to-end database execution times, which can be slow. Furthermore, a recalculation is necessary for every database and hardware change. For these reasons, we consider improving the underlying cost model to output more accurate end-to-end runtime predictions as the critical challenge.

**Objective O3: Dollar-Budget Constrained Buying Recommendations** The third objective serves the use case that a user has a certain amount of money to spend on their infrastructure. While allocating more central processing unit (CPU) resources can be a strategy to reduce query runtimes, we focus on a given budget for memory and storage devices. With their monetary budget  $m$ , the user wants to buy the devices that allow for the best runtime performance. The parameter  $g_d$  stores the cost of a device  $d$  in dollars per byte.

$$\text{minimize}_{x_{t,a,p,d} \in \{0,1\}^{T \times A \times P \times D}} \sum_{\substack{t \in T, a \in A, \\ p \in P, d \in D}} x_{t,a,p,d} \cdot c_{t,a,p,d} \quad (7)$$

$$\text{s.t.} \quad \left( \sum_{\substack{t \in T, a \in A, \\ p \in P, d \in D}} x_{t,a,p,d} \cdot s_{t,a,p} \cdot g_d \right) \leq m \quad (8)$$

$$\left( \sum_{d \in D} x_{t,a,p,d} \right) = i_{t,a,p} \quad \forall t \in T, a \in A, p \in P \quad (9)$$

The objective (7) of this ILP model is to minimize the predicted runtime costs of the data placement. The objective is subject to two constraints. The constraint (8) asserts that the data assignment to the devices does not exceed the monetary budget. Furthermore, this ILP model also includes constraint (9) that forces segments to be assigned to exactly one device.

In the above model for objective O3, we considered the dollar costs of the devices as a continuous price in dollars per used byte. This assumption can hold for fine-granular storage rentals from cloud service providers. However, we are limited to a discrete set of available byte capacities when purchasing raw storage device hardware. To support discrete device capacities, we replace the constraint (8) with the constraint (11). For each device  $d$ , we assume a given list of length  $J$  that is sorted in ascending order. The list contains the discrete device capacities  $e_{d,j}$  where  $j = 1..J$  indexes the  $J$  distinct available capacities. Furthermore, equation (10) introduces a variable  $s_{x,d}$  for the size in bytes of the segments assigned to a device  $d$  according to the values of the decision variables  $x_{t,a,p,d}$ .

$$s_{x,d} = \sum_{\substack{t \in T, a \in A, \\ p \in P}} x_{t,a,p,d} \cdot s_{t,a,p} \quad (10)$$

$$\left( \sum_{d \in D} g_d \cdot \text{argmin}_{e_{d,j}, s_{x,d} \leq e_{d,j}} e_{d,j} \right) \leq m \quad (11)$$

The formalization above contains a non-linearity in the *argmin* expression (11) as it is not a linear combination of its input variables. Therefore, we cannot solve this model in the presented form using linear solvers. However, the non-linearity can be substituted by an equivalent formulation in linear terms.

**Evaluation Setup** We conduct our measurements on a machine with two AMD EPYC 7F72 CPUs, each with 24 physical cores, 48 threads, a 192 MB shared L3 cache, and 256 GB of DDR4 memory with a theoretical per-socket memory bandwidth of 204.8 GB/s [Ad21]. Per CPU, the DRAM is distributed across eight Samsung M393A4G43AB3 dual in-line memory modules (DIMMs), each with a size of 32 GB. We pin processes and memory to a single node using `numactl` for our measurements on this multi-socket system. The machine

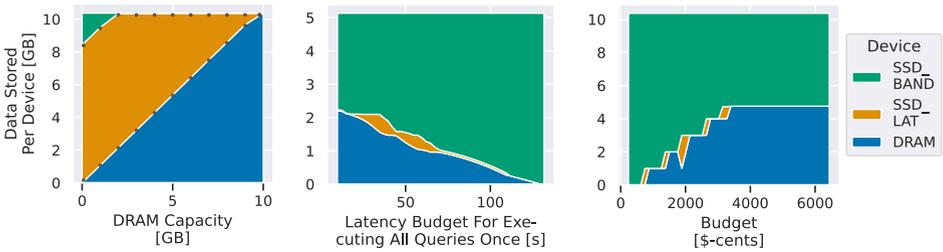
runs Ubuntu 22.04 LTS with the Linux kernel 5.15.0-41-generic. We compile Hyrise with GCC 10.3 and use Python 3.10.4 to execute the placement selection module. We configure Hyrise to use all available cores on the given non-uniform memory access (NUMA) node.

For our measurements, we set the number of cores that Hyrise can use to 24, which is the number of available physical cores on one NUMA node of our test system. Furthermore, we set the number of clients to eight. Each client corresponds to one stream of queries we send to the database system concurrently. Moreover, we randomize the order of the queries for each client to utilize the database’s resources evenly. In combination, our parameter values for the number of cores and clients allow us to measure the multi-threaded database performance. To evaluate the query latencies, we execute queries of the join order benchmark (JOB), TPC Benchmark DS (TPC-DS), and TPC Benchmark H (TPC-H) benchmarks. Unless specified otherwise, we set the scale factors of TPC-H and TPC-DS to ten to obtain a data size that exceeds the benchmark machine’s cache sizes. The JOB does not have a scale factor.

In our experiments, we use the following three devices: DRAM, a redundant array of independent disks (RAID) of two Micron 7450 NVMe Express (NVMe) SSDs (*SSD\_BAND*), and a low-latency Intel Optane DC Series SSD (*SSD\_LAT*). The SSD RAID is of type 0. *SSD\_LAT* has a lower access latency, while *SSD\_BAND* offers higher bandwidth. In measurements with the `fiio` utility [Ax22] and the Intel Memory Latency Checker (MLC) [Vi21], the devices DRAM, *SSD\_BAND*, and *SSD\_LAT* had a read latency of 130, 70 000, and 12 000 nanoseconds, respectively. Furthermore, the sequential multi-core read bandwidth was 141.3, 12.5, and 2.4 GB per second. As a baseline, we also investigated a second set of devices consisting of DRAM, an SSD, and an HDD. We refer to this set as the *ordered device set*, opposed to the *tierless device set*. These devices have a clear ordering by their access performance.

We use the commercially available Gurobi solver [Gu21a] to solve the specified LP models. Previous research has found this solver to offer good runtime performance for similar applications [Bo22, p. 785] compared to other solvers, such as SCIP [Ga20] or Cbc [Lo03]. The optimality gap [Gu21b] is the maximum accepted gap between the objective value of the solution that the solver terminates with and the optimal objective value. We set this optimality gap to 1% as we experienced solver timeouts for smaller values. Furthermore, we limit the maximum execution time to 500 seconds. Finally, we set the number of threads to the number of cores of the test machine, i.e., 24 threads. We formulate the LP models using Pyomo [By21, HWW11], a Python-based open-source optimization modeling language that allows for interchangeable solvers.

**Exemplary Placement Decisions** Fig. 2 shows exemplary placement decisions of our system for the TPC-H benchmark using the three objectives O1, O2, and O3. The placement behavior for the JOB and TPC-DS benchmark shows similar patterns. For each given constraint (e.g., DRAM capacity, latency budget, dollar budget) that we vary along the x-axis, the plots display the percentage of data stored on the respective devices on the y-axis. Fig. 2a shows the objective O1 placements. For each DRAM budget, the algorithm uses all available DRAM as it allows for the lowest predicted workload runtime. We define the



(a) O1 device usages. The DRAM budget is varied between zero and ten GB in 11 steps. (b) O2 purchase recommendations. The latency budget is varied between 9 and 132 seconds in 50 steps. (c) O3 purchase recommendations for discrete device sizes. The dollar budget is varied between 241 and 6436 dollars in 50 steps.

Fig. 2: Exemplary Data Placements for objective O1, O2, and O3 for the TPC-H benchmark with scale factor ten. TPC-H scale factor five for the O2 purchase recommendations.

workload runtime as the runtime required to execute all queries of a workload (e.g., all TPC-H queries) once. For a DRAM budget of zero GB, most segments are assigned to SSD\_LAT while SSD\_BAND holds two GB of data. The latency-optimized SSD\_LAT holds the segments with predominately random accesses, while SSD\_BAND holds segments that are frequently accessed sequentially. In a comparison between the tierless device set and the alternative ordered device set, we found that our placement system can leverage the tierless property adequately. Fig. 2b shows the placements for objective O2. For the minimum latency budget, approximately 45 percent of the data is stored in DRAM. In comparison, the remaining 55 percent of data are unused segments that can be stored on SSD\_BAND without affecting the workload latency. For the JOB and TPC-DS benchmark, 25 and 49 percent of the data is unused, respectively. In our algorithm, a post processing step assigns the unused segments to the least expensive device with available capacity. Directly implementing this functionality in the LP model by adding a device penalty to the unused segments' cost proved infeasible. The device penalty value had to be larger than the optimality gap but smaller than the minimum cost for used segments. With the minimum cost for used segments being smaller than the optimality gap in our experiments, this was not possible. Fig. 2c shows the objective O3 placements for discrete device sizes. With an increasing dollar budget, the algorithm can gradually afford more device space for faster devices, such as DRAM. As we artificially limited the available discrete device capacities to integer GB values (e.g., 1 GB, 2 GB, 3 GB), the algorithm affords the devices in steps. The less expensive SSD\_LAT precedes the DRAM purchase. Interestingly, the DRAM usage is not monotonically increasing. For a dollar budget of 2000 cents, the model increases the SSD\_LAT usage to two GB while it reduces the DRAM usage from two GB to one GB. Random access is the dominant access pattern of the segments assigned to SSD\_LAT. As the bandwidth-optimized SSD\_BAND has a higher read latency than the other devices, this decrease in DRAM usage thus allows the model to assign more random access-heavy segments to other devices than SSD\_BAND and minimize the predicted runtime.

### 2.3 Dynamic Workloads

We define a workload as dynamic if it has a temporal skew in the queries being run during its duration. Our system supports these workloads by updating the placement regularly. We use windowing to collect access tracking information. The system bases the placement decisions on only the tracked segment accesses that occurred since the last placement update. In our experiments, we set the window size to two minutes. With these periodic updates, we successfully adapted the placement to the changing workload and reduced the query latencies accordingly. However, this reactive approach cannot predict future workloads. Furthermore, recurring workload changes might cause the placement to oscillate between multiple configurations, incurring high segment migration costs. Frequent and fast updates of the data placement are critical to quickly react to workload changes. The update frequency is limited by the runtime of the placement algorithm and the runtime required to apply a placement configuration. Thus, dynamic workload support is an application that requires low-latency placement algorithms, which we investigate in Sect. 4.2.

## 3 Data Placement Cost Models

We want to investigate how to efficiently estimate data placement effects on query runtime with sufficient accuracy using simple cost models. For this, we build cost models based on (i) Hyrise’s segment access counters and (ii) device calibration data.

### 3.1 Assumptions

All cost models proposed in this work make the following similar underlying assumptions.

**I/O-Dominated Workloads** Our approach focuses on estimating data access costs. Thus, we do not directly estimate the costs of operators, such as joins or aggregates, executed during query processing. However, we indirectly include their costs as these operators might perform data accesses tracked with Hyrise’s access counters. Similarly to Vogel et al. [Vo20, p. 2666], we argue that our focus on data access costs might decrease the absolute accuracy of our runtime predictions. However, the runtime predictions can still be correct in relation to each other. Nevertheless, even this relative measure is subject to the accuracy of the data access runtime cost predictions. As an example for input/output (I/O)-focused runtime predictions, let us consider a workload defined by one query that is executed. For this query, we assume that the CPU-heavy work in the query’s operators (e.g., building a hash map in a join operator) is independent of the data access in the operators (e.g., materializing all values of a position list before building the hash map). The CPU-heavy work then adds a static overhead that is independent of the data placement of the segments. Therefore, we can compare the cost estimations of different data placements, and the differences between the estimates are correct, except for a constant overhead. In general, I/O is often a bottleneck for modern CPUs [HSY01]. Thus, I/O-dominated workloads are a relevant category of workloads. We argue that data access costs can be a good approximation for these workloads’ overall query latency performance with the previous reasoning.

**Independent Data Placement Decisions** In this work, a placement decision for one segment can be made independently of a placement decision for another segment. While interactions between placement decisions could be possible, we exclude this case to limit the complexity of the placement algorithms. Similarly, our cost models make independent predictions per segment. This simplification allows us to limit the complexity of the cost models but can lead to inaccuracies. For example, an operator execution might read two columns in parallel, and its execution time might be determined by the maximum scan time of both columns. Furthermore, we consider the devices’ access performance characteristics unaffected by our placement decisions. Similarly to the simplification above, this assumption allows us to limit the complexity of the data placement algorithms and cost models. To illustrate where this assumption can fail, let us assume that an operator execution reads two segments in parallel. If these two segments reside on the same device, the device’s bandwidth can be impacted, and it could be beneficial to distribute the segments to different devices. On HDDs, multi-threaded reads can even decrease the read throughput due to increased seek time [Vo20, p. 2666].

### 3.2 Cost Model Definition

We propose the calibrated and workload-based cost model C3. Furthermore, we compare model C3 with previous development iterations C0, C1, and C2. The cost model C3 uses device calibration data and segment access information to estimate the runtime performance impact when a segment is assigned to a specific device.

$$c_{t,a,p,d}^3 = \sum_{\gamma \in \Gamma} u_{d,\gamma,\xi_{t,a,p}} \cdot h_{t,a,p,\gamma} \cdot \frac{s_{t,a,p}}{n_{t,a,p}} \quad (12)$$

Model C3 predicts the runtime cost of assigning a segment in table  $t$ , attribute  $a$ , and partition  $p$  to device  $d$  using the formula in (12). The formula sums over all access patterns  $\gamma \in \Gamma$  (sequential, monotonic, random, and point). Per access pattern, the model calculates the cost prediction as a product of the calibration value  $u_{d,\gamma,\xi_{t,a,p}}$ , the access counter  $h_{t,a,p,\gamma}$  for the respective access pattern, and the byte size per value  $\frac{s_{t,a,p}}{n_{t,a,p}}$ , where  $n_{t,a,p}$  is the number of values in a segment. The calibration value  $u$  takes the parameter  $\xi_{t,a,p}$  that yields the segment’s data type. The set of possible values  $\xi_{t,a,p} \in \Xi = \{string\_SSO, float\}$  contains two data types: non-small string optimization (SSO) strings and floating-point numbers. Our specialized *string-SSO* calibration reads strings with an average length of 44 bytes, which exceeds the SSO threshold. The *float* calibration is used as the default for all other segment data types. SSO is a technique that compilers use to avoid dynamic memory allocations on the heap and improve data locality. As Hyrise stores the values of string segments as C++ `std::string` objects, this optimization applies in Hyrise. If a string’s size is below the SSO threshold, the content of the string can be stored on the stack in the string object itself. For example, GCC has an SSO threshold of 15 bytes. Contrarily, the content of strings that exceed this threshold is stored on the heap, which means that the string object has to hold a heap pointer to the underlying string buffer. This additional pointer redirect when reading the string’s content is equivalent to random access, as the

memory layout of the underlying string buffers is undefined. For strings residing on devices with poor random access characteristics (i.e., a high read latency), access to string segments can incur high runtime costs. We experimentally determine the calibration values  $u_{d,\gamma,\xi_{t,a,p}}$  by benchmarking a read workload for each combination of access pattern, device, and data type. For the data type, we consider strings with a length larger than 15 bytes and floating-point numbers with single precision. However, segments of integer data type showed the same calibration values in our experiments. The calibration workload reads all values of a column with an uncompressed total size of 720 MB. Google Benchmark [Go22a] repeats the measurements until a stable runtime is reached. Between benchmark iterations, random data is read to flush the UMap cache and the CPU caches, so we do not measure cache effects. The calibration finishes in less than 90 minutes, though it could be reduced to a fraction of that by reducing the number of values read without significantly sacrificing accuracy.

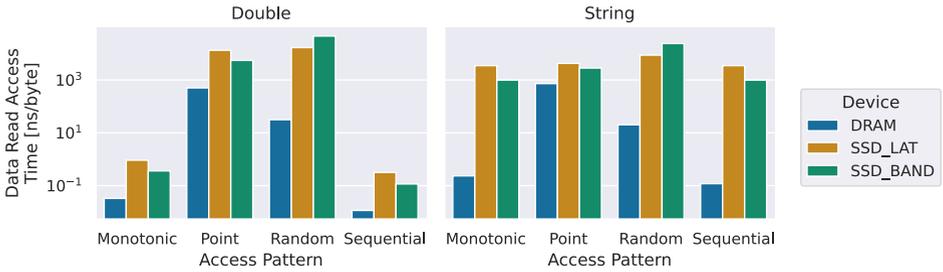


Fig. 3: Device calibration for different access patterns, data types, and devices. Multi-threaded reads (24 threads) measured on TPC-H data with scale factor ten.

Fig. 3 shows the calibrated runtimes for both data types. The calibrated values are the read times normalized as nanoseconds to read one byte with the respective access pattern. For example, the calibrated sequential byte access time for DRAM is 0.011 nanoseconds, equivalent to a read bandwidth of 90 GB per second. This calibrated DRAM bandwidth is smaller than the maximum bandwidth measured with Intel MLC of 141.3 GB per second, demonstrating that the calibration can improve the cost model’s accuracy. We explain this difference between theoretical and utilized bandwidth with CPU overhead for decoding and processing the read data. In the calibration data, the low-latency SSD\_LAT shows faster random access speed while SSD\_BAND is faster for the remaining access patterns. For example, for the sequential accesses, the SSD\_BAND achieves a bandwidth of 8.25 GB per second, which exceeds the theoretical maximum bandwidth of SSD\_LAT by 3.4 times. The calibration not only allows us to distinguish the runtime performance of the devices but also supplies information about the importance of specific access patterns and data types. Fig. 3 shows that sequential access allows for the fastest read speed while the random access pattern has the highest runtime. Previous research showed that the ability to pre-fetch and cache data significantly influences the runtime performance of random data accesses [He21, p. 32]. Therefore, data accesses that follow the random and point access pattern incur the highest runtime. However, in an exemplary TPC-H benchmark run in

Hyrise,  $10^6$  as many random accesses as point accesses were recorded. These measured calibration values thus support the findings of Dreseler [Dr22, p. 120] that random access runtime performance greatly influences end-to-end database system runtime for Hyrise. The calibrated runtime for the string data type is significantly higher than for the *float* data type. The geometric mean of the string calibration values is 389.6 nanoseconds, which is 20 times higher than 19.9 nanoseconds, the geometric mean of the *float* calibration. We explain these differences with the second pointer indirection required to read non-SSO strings, which is equivalent to random memory access.

We compare the presented cost model with three iterations shown in (13)-(15). Model C0 does not use calibration data and corresponds to the model proposed by Dreseler [Dr22] using manually-determined weights  $w_\gamma$ . The superscript number  $\nu$  in the cost formula  $c_{t,a,p,d}^\nu$  indicates the cost model version.

$$c_{t,a,p,d}^0 = \sum_{\gamma \in \Gamma} w_\gamma \cdot h_{t,a,p,\gamma} \quad (13)$$

$$c_{t,a,p,d}^1 = \sum_{\gamma \in \Gamma} u_{d,\gamma} \cdot h_{t,a,p,\gamma} \quad (14)$$

$$c_{t,a,p,d}^2 = \sum_{\gamma \in \Gamma} u_{d,\gamma} \cdot h_{t,a,p,\gamma} \cdot \frac{s_{t,a,p}}{n_{t,a,p}} \quad (15)$$

Furthermore, we considered extending our cost model C3 with a cache miss rate prediction, as introduced by Lasch et al. [La22]. However, the proposed function did not model the CPU cache behavior accurately in our experiments. The maximum segment size in Hyrise was one order of magnitude smaller than the last level cache size, and thus the predicted cache miss rate was 0.05 for all segments. Therefore, the authors' assumption that the cache miss rate is independent for each single data structure was not met in our application.

### 3.3 Evaluation

The upper plots in Fig. 4 show the measured workload runtime for data placements determined with the objective O1 algorithm based on the respective cost model. The runtimes were measured for the JOB, TPC-DS, and TPC-H benchmark with scale factor ten and the DRAM capacity is varied between zero GB and full capacity. We consider only DRAM and SSD\_BAND due to the limitations of cost model C0. The data placements for zero GB DRAM capacity and full capacity are thus the same across all cost models because all segments are assigned to the same device. Consequently, the measured query latencies are almost equal for these two cases. For all three workloads, the figures show that DRAM capacity thresholds exist where adding additional capacity does not decrease the end-to-end runtime. These thresholds correspond to the unused data per benchmark. Compared to the C3 cost model, the mean end-to-end measured runtimes for the C0, C1, and C2 models are 31, 18, and 3 percent higher, respectively. The lower plots in Fig. 4 show the predicted runtimes by the respective cost models. The cost models predict the

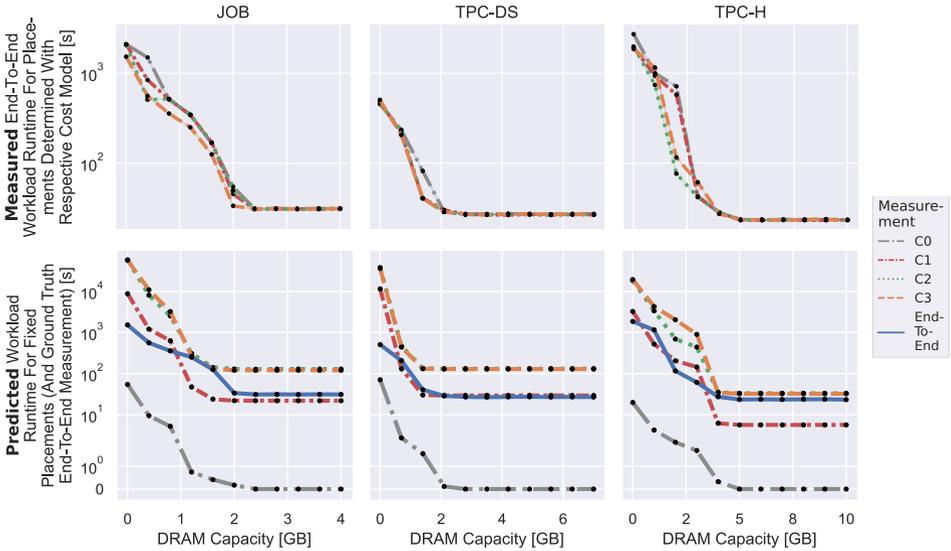


Fig. 4: Comparison of (i) measured end-to-end workload runtime and (ii) runtime prediction accuracy for data placements determined with the objective O1 ILP algorithm based on the respective cost model.

workload runtimes for data placements determined with the O1 algorithm and cost model C3. Model C0 underestimates the runtime by up to two orders of magnitude compared to the end-to-end measured runtimes. We argue that the costs predicted by C0 are on an arbitrary scale as it is not calibrated and cannot be seen as runtime predictions. Model C3 often overestimates the workload runtime. Potential reasons why the end-to-end measured runtimes are lower than expected are that model C3 overestimates data access costs because parallelization and caching speed up data accesses.

In conclusion, we showed that calibrated cost models offer significantly better cost predictions than models with manually-determined weights. To efficiently estimate data placement effects on runtime, we argue that our calibrated cost models are both simple and offer sufficient accuracy for our application. Assessing the accuracy of single cost predictions is challenging because no *gold standard* cost model exists to compare against. Experimentally determining the cost of each segment placement is infeasible due to the large size of the placement decision space. However, the demonstrated end-to-end placement decision and the runtime prediction accuracy of cost model C3 show solid results and allow us to recognize differences between the placement selection algorithms. Further adjustments to the device calibration (e.g., by better modeling the caching behavior) could improve the runtime prediction accuracy. Based on our development efforts, we argue that both (i) the usage of calibration data and (ii) the calibration data quality are significant factors for making calibrated cost models a reliable decision basis for placement selection algorithms.

## 4 Placement Selection Algorithms

Previous research [Bo22] has shown that simple heuristics can be tractable alternatives to optimal solutions because they often offer comparable results at lower algorithm runtime and memory usage. We investigate which trade-offs users have to make between different data placement algorithms. We compare algorithms for objective O1 as a proxy for all objectives. The algorithms include the LP algorithm introduced in Sect. 2, a second linear programming algorithm that makes placement decisions with column granularity, a greedy heuristic, and a multi-tier Knapsack algorithm.

### 4.1 Algorithm Descriptions

In the Greedy and Knapsack algorithm, we require computing an access performance metric  $v_d$  per device  $d$  and calculate it with the formula in (16). The existence of such an access performance metric implies an ordering of the devices by their access speeds (e.g., latency, bandwidth). This ordering contradicts our goal to organize the memory and storage mediums in a tierless device pool with no fixed ordering. However, this sorting is not used in our ILP solution.

$$v_d = \sum_{\gamma \in \Gamma} \left( \sum_{\xi \in \Xi} u_{d,\gamma,\xi} \cdot \sum_{t \in T, a \in A, p \in P} h_{t,a,p,\gamma} \right) \quad (16)$$

**Multi-Device Greedy** The Greedy algorithm orders the devices by their access performance and assigns the segments greedily to the fastest devices. This algorithm bases on the greedy heuristics proposed by Boissier et al. [BSU18, p. 214] and the *HOT* strategy by Vogel et al. [Vo20, pp. 2667–2668]. These two related works use cost models optimized for their respective database systems and intertwine the cost models with the placement selection algorithms. In our work, we focus on the Greedy algorithm itself and distinguish the placement selection algorithm from the cost model.

In the algorithm described in Algorithm 1, we first sort the devices by their access performance  $v_d$  in ascending order. To assign segments to devices, we regard the devices in a pairwise manner, starting with the fastest devices. We thus model the decision problem as a series of binary decisions between two devices. For each pair, we compute in Line 6 the segment scores and sort the segments according to these scores in descending order. A segment’s score is calculated as the difference between the segment’s cost values for the two corresponding devices. The variable  $c_{w,d}$  holds the predicted costs for assigning a segment  $w$  to device  $d$ . We then greedily assign the segments to the current device as the device’s byte capacity permits. Assuming  $S$  is the number of segments and  $D$  the number of devices, the asymptotic complexity of this Greedy algorithm is  $O(S \cdot \log S \cdot D + D \log D)$ . However, the number of devices  $D$  is constantly three in our experiments. Another difference between our Greedy algorithm and the previously-mentioned greedy algorithms from related work is our focus on supporting more than two devices. For example, the *HOT* algorithm at table granularity by Vogel et al. “places tables descending in order of their number of accesses on the fastest device with enough space for the whole table” [Vo20, p. 2668]. In comparison

**Algorithm 1:** Greedy Placement Selection Algorithm

**Data:** set of all segments  $W$ , segment sizes, segment costs  $c_{w,d}$ , device byte capacities, device calibration

**Result:** data placement

```

1 sorted devices = sort devices by  $v_d$  ascending;
2 for  $(d_i, d_{i+1})$  in sorted devices do
3   if all segments assigned then
4     | return data placement;
5   end
6   sorted segments = sort segments  $w \in W$  that are unassigned by  $(c_{w,d_{i+1}} - c_{w,d_i})$  descending;
7   for segment  $w$  of sorted segments do
8     | if  $w$  fits onto device  $d_i$  then
9       | | assign  $w$  to  $d_i$ ;
10    | end
11  end
12 end

```

with these related work algorithms, we argue that our Greedy algorithm can produce data placements for multiple devices that induce lower end-to-end query latencies because we model the cost increase between the two devices instead of the absolute costs. With the proposed technique, our Greedy algorithm resembles the cost model usage of our ILP models.

**Linear Programming with Column Granularity** We compare the objective O1 LP algorithm and an adapted version that uses columns instead of segments as the placement decision unit. We refer to this segment-granular LP algorithm as *LP* and the algorithm that makes decisions at a column granularity as *Column-LP*. In related work by Vogel et al. [Vo20, p. 2674], the authors of the Mosaic storage engine find that their column-granular *HOT column* strategy outperforms the table-granular *HOT table* algorithm by a factor of 1.99. Similarly, we compare column and segment granularities. Our Column-LP is inspired by Mosaic’s LOPT optimization model, which also uses column granularity. A complete re-implementation of the LOPT model was infeasible for our comparison because the LOPT model builds on a cost model that is intertwined with the authors’ proposed LP model and specific to the database system’s execution model.

**Multi-Tier Knapsack** The multi-tier Knapsack algorithm is an implementation of the multilevel generalized assignment problem (MGAP) simplification proposed by Dresler [Dr22, p. 113]. The author models the placement selection problem as a series of independent binary decision problems between pairs of devices, similarly to our proposed Greedy algorithms in Sect. 4.1. Due to this simplification, the MGAP approach does not necessarily yield the optimal solution. The algorithm first sorts the devices by their access performance  $v_d$  in ascending order. The devices are regarded in a pairwise manner to assign the segments to them, starting with the fastest devices. Each binary decision problem is formulated as a Knapsack problem to determine which segments to place on the current device  $d_i$ . All

segments not selected for the current device will be reconsidered for the next device pair. Thus, we do not assign segments to device  $d_{i+1}$ . The segments correspond to the Knapsack items. The computed segment costs  $c_{w,d_{i+1}} - c_{w,d_i}$  are the items' values, and the segment sizes are the items' weights. The byte capacity of the current device defines the size of the Knapsack. We use the Google OR-Tools `KNAPSACK_MULTIDIMENSION_BRANCH_AND_BOUND_SOLVER` with a timeout of 500 seconds [Go22b].

## 4.2 Evaluation

The upper plots in Fig. 5 show the predicted workload runtime for data placements determined with the respective algorithm. The LP algorithm consistently determines the placements with the lowest runtime predicted by cost model C3. Relative to the LP algorithm's mean predicted runtime, the Column-LP algorithm's placements incur the highest mean predicted runtime with 203% of the LP placement's runtime. The Column-LP is followed by the Greedy algorithm at 136% and the Knapsack algorithm at 115% of the LP placement's runtime. The column-granular decisions of the Column-LP incur significantly higher runtimes because (i) they cannot capture vertical data access skew and (ii) they cannot exhaust the device capacities as efficiently as segment-granular placements. Furthermore, the Knapsack and Greedy algorithms are unable to determine the optimum data placement in some conditions due to their greedy strategy.

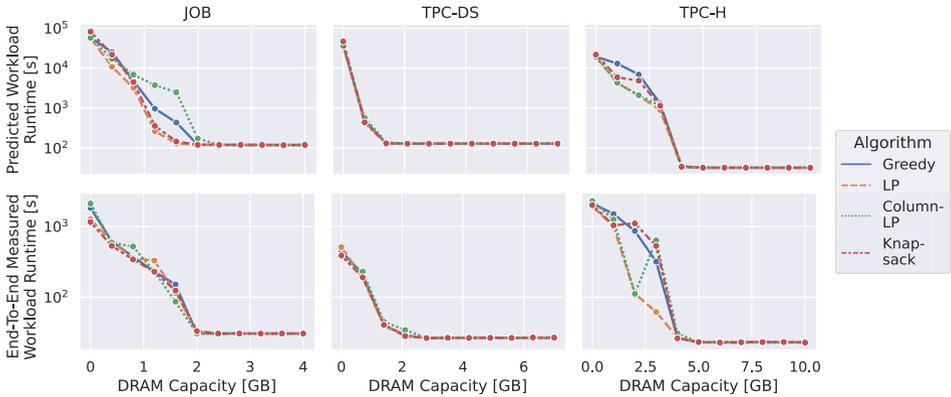


Fig. 5: Predicted and measured runtime of data placements determined with different placement algorithms based on cost model C3. Measurements for JOB, TPC-DS with scale factor ten, and TPC-H with scale factor ten.

The lower plots in Fig. 5 show the end-to-end measured workload runtime for data placements determined with the respective algorithm. Due to cost prediction inaccuracies, not all measured runtimes show the same patterns as the predicted runtimes. In our experiments, we also discovered cases where simpler algorithms (e.g., Greedy, Knapsack) produced placements that incurred lower end-to-end runtimes than the placement determined by the LP algorithm because of these inaccuracies. However, on average, the LP algorithm

determines the placements with the lowest end-to-end workload runtime. The Knapsack algorithm’s results have the highest mean predicted runtime at 147 percent of the LP algorithm’s results. The Greedy algorithm follows at 135 percent and the Column-LP algorithm at 132 percent.

An evaluation of the trade-off between algorithm runtime and solution quality is shown in Fig. 6. The algorithm choice is subject to a pareto-optimal trade-off between both metrics. Compared to the LP algorithm, the Knapsack algorithm’s resulting data placements are only 15 percent less optimal, while the algorithm has an 80 percent shorter runtime. Similarly, the Greedy algorithm trades a 36 percent optimality decrease for an 81 percent runtime decrease, making it a viable alternative to the optimal LP algorithm. For example, for the TPC-H benchmark with scale factor 1 000, 1 818 000 segments need to be assigned to the devices. In this context, the LP algorithm takes 60 minutes to determine a data placement, whereas the Pyomo setup time and the Gurobi solver runtime are responsible for one-third of the runtime, respectively. In contrast, the Greedy and Knapsack algorithms terminate within 20 minutes, and the Column-LP algorithm takes only 81 seconds. Memory consumption measurements show similar patterns.

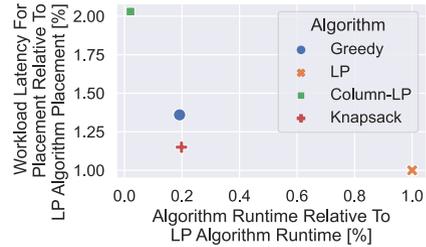


Fig. 6: Mean algorithm runtime and solution quality across JOB, TPC-DS, and TPC-H benchmarks. The solution quality corresponds to the cost model C3’s predicted runtime to execute the workload queries once per data placement.

## 5 Related Work

Several related works on automated decision-making are shown in Tab. 1. Some of the stated research does not determine data placements but related configurations, such as encoding or index selection. Similar to our approach, the Mosaic storage system for the Umbra DBMS [Vo20] determines data placements with a linear optimization model and two greedy algorithms. However, Mosaic uses a cost model focused on sequential reads, matching Umbra’s data access patterns. The authors compare column-granular data placements with table-granular placements and find a 1.99× relative speedup. Further related works include the Hybrid Data Layouts for Tiered HTAP Databases [BSU18] and the automatic tiering research by Dresler [Dr22].

Boissier [Bo22] proposes an optimal linear programming encoding selection algorithm and a greedy heuristic that he often found on par with the optimal solution. The heuristic strategy weighs the candidates by their benefit-to-cost ratio [Va00]. To predict runtime costs of encoding configurations, Boissier uses multiple linear regression models comparable to the work of Ma et al. [Ma21]. Another cost model approach are zero-shot models [HB22].

| Related Decision-Making Work  | Granularity     | Config Optimization |          |       |                | Placement Features |              |              |              |
|-------------------------------|-----------------|---------------------|----------|-------|----------------|--------------------|--------------|--------------|--------------|
|                               |                 | Data Placement      | Encoding | Index | Storage Layout | Device Count       | Objective O1 | Objective O2 | Objective O3 |
| Mosaic [Vo20]                 | Columns         | ●                   | ◐        | ○     | ○              | n                  | ●            | ○            | ●            |
| Hybrid Layout Hyrise [BSU18]  | Columns         | ●                   | ○        | ○     | ○              | 2                  | ●            | ○            | ○            |
| Automatic Tiering [Dr22]      | Segments        | ●                   | ○        | ○     | ○              | 2                  | ●            | ○            | ○            |
| Encoding Configuration [Bo22] | Segments        | ○                   | ●        | ○     | ○              | -                  | -            | -            | -            |
| Config Optimization [RSB22]   | Segments        | ●                   | ●        | ●     | ○              | n                  | ●            | ○            | ○            |
| Cost Modeling HANA [La22]     | Data Structures | ●                   | ○        | ○     | ○              | 2                  | ●            | ○            | ○            |
| Proteus and Tiresias [ALD22]  | Varying         | ●                   | ●        | ●     | ●              | 2                  | ●            | ○            | ○            |
| This Work                     | Segments        | ●                   | ○        | ○     | ○              | n                  | ●            | ●            | ●            |

Tab. 1: Overview of Related DBMS Configuration Selection Research

However, compared to our calibrated cost models, such learned models have high complexity, long training times, and are hard to generalize.

Richly et al. [RSB22] optimize multiple configuration aspects jointly. The underlying cost model exhaustively calibrates scan operations for various configurations, which results in a long preparation phase to establish cost estimates.

Lasch et al. [La22] propose a data placement cost model for PMem, limiting the number of devices where a data unit can be placed to two (DRAM and PMem). Their model uses (i) device calibration data and (ii) workload information in the form of access counts, similar to our model C3. However, the authors also model the costs for additional data structures and regard cache miss ratios.

Finally, Abebe et al. [ALD22] propose Tiresias, a storage cost model that can predict future workloads to optimize multiple configuration aspects jointly. Such a predictive capability could be useful for our data placement system to anticipate dynamic workload changes.

## 6 Discussion

We proposed an automatic placement selection system for IMDBMS that supports multiple placement objectives, makes workload-driven placement decisions, and manages its devices in a tierless pool. In our exemplary implementation for the database system Hyrise, we compared several cost models and placement selection algorithms. The proposed cost models are applicable for database systems making similar assumptions as Hyrise (cf. Sect. 3.1), while the proposed placement algorithms have general applicability.

Based on our comparison of an optimal linear programming algorithm, a heuristic based on the Knapsack problem, and a greedy algorithm, we argue that the algorithm choice is

subject to a pareto-optimal trade-off between algorithm resource usage and solution quality. The Knapsack and Greedy algorithms are viable alternatives to the resource-intensive LP algorithm, especially under inaccurate cost predictions. Placement granularity significantly influences the solution’s optimality, as we found column-granular placements to incur, on average, 103 percent higher query latencies than segment-granular decisions. However, such column-granular decisions can be viable for low-latency applications (e.g., frequent placement updates for dynamic workloads) requiring fast re-computations of the data placement, as the column-granular linear programming algorithm required up to two orders of magnitude less runtime compared to the segment-granular LP algorithm.

We found our cost model using (i) data access pattern tracking information and (ii) device calibration data to offer sufficient accuracy to distinguish the differences between placement selection algorithms and determine suitable placements. While other approaches, such as learned cost models, can offer higher accuracy, they can be complex, hard to generalize, and require expensive data collection. In comparison, our cost model is inexpensive to calibrate.

Merely moving the unused data from DRAM to secondary devices already allows for significant DRAM usage reductions. For the JOB, TPC-DS, and TPC-H benchmark, the share of data that could be removed from DRAM without increasing the workload latencies were 25, 49, and 55 percent, respectively. Even naive algorithms and cost models can determine such placement decisions, as it suffices to track data accesses.

**Future Work** To further improve our system’s data placements and determine the optimal end-to-end placement, the choice of both the placement algorithm and the cost model is relevant. Placement selection algorithms determining placements close to the optimal solution are a necessary condition. In comparison, an optimal cost model accurately predicting all real-world database system behavior does not exist. Thus, placement cost modeling is a complex challenge that requires further research. Future improvements to our cost model include calibration for all available data types and predicting database system behavior such as caching. Additionally, operator-granular placement cost models could enable robustness guarantees for single query runtimes. Furthermore, an open question is whether the Greedy placement algorithm could be improved with alternative segment sorting metrics (e.g., by their benefit-to-cost ratio). In addition, we consider extending objective O3 to optimize purchases of the entire infrastructure, including CPU resources, a future work item. In this work, we focused on block-level devices as placement alternatives to the system’s DRAM. Extending our system to place data on heterogeneous memory with different access qualities, e.g., CPU-local DRAM and Compute Express Link (CXL)-attached [CX22], disaggregated memory, is a potential future effort.

## References

- [Ad21] Advanced Micro Devices, Inc: AMD EPYC 7F72 - Technical Specification, <https://www.amd.com/en/product/9656>, 2021, visited on: 10/05/2022.
- [ALD22] Abebe, M.; Lazu, H.; Daudjee, K.: Tiresias: Enabling Predictive Autonomous Storage and Indexing. *Proc. VLDB Endow.* 15/11, pp. 3126–3136, 2022.
- [Am21] Amazon Web Services, Inc.: Maximum Transfer Speed between Amazon EC2 and Amazon S3, <https://aws.amazon.com/premiumsupport/knowledge-center/s3-maximum-transfer-speed-ec2/>, 2021, visited on: 10/05/2022.
- [Ap19] Appuswamy, R.; Graefe, G.; Borovica-Gajic, R.; Ailamaki, A.: The Five-Minute Rule 30 Years Later and Its Impact on the Storage Hierarchy. *Commun. ACM* 62/11, pp. 114–120, 2019.
- [Ax22] Axboe, J.: Flexible I/O Tester, <https://github.com/axboe/fio>, 2022, visited on: 10/04/2022.
- [Bo22] Boissier, M.: Robust and Budget-Constrained Encoding Configurations for In-Memory Database Systems. *Proc. VLDB Endow.* 15/4, pp. 780–793, 2022.
- [BSU18] Boissier, M.; Schlosser, R.; Uflacker, M.: Hybrid Data Layouts for Tiered HTAP Databases with Pareto-Optimal Data Placements. In: *Proceedings of the IEEE International Conference on Data Engineering, ICDE*. Pp. 209–220, 2018.
- [By21] Bynum, M. L.; Hackebeil, G. A.; Hart, W. E.; Laird, C. D.; Nicholson, B. L.; Siirola, J. D.; Watson, J.-P.; Woodruff, D. L.: *Pyomo—Optimization Modeling in Python*. Springer Science & Business Media, 2021.
- [CX22] CXL Consortium: Compute Express Link: The Breakthrough CPU-to-Device Interconnect, <https://www.computeexpresslink.org>, 2022, visited on: 10/09/2022.
- [Da21] Daase, B.; Bollmeier, L. J.; Benson, L.; Rabl, T.: Maximizing Persistent Memory Bandwidth Utilization for OLAP Workloads. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Pp. 339–351, 2021.
- [Dr19] Dreseler, M.; Kossmann, J.; Boissier, M.; Klauck, S.; Uflacker, M.; Plattner, H.: Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management. In: *Proceedings of the International Conference on Extending Database Technology, EDBT*. Pp. 313–324, 2019.
- [Dr22] Dreseler, M.: Automatic Tiering for In-Memory Database Systems, DOI: 10.25932/publishup-55825, PhD thesis, Universität Potsdam, 2022, 143 pp.
- [Du16] Dulloor, S.; Roy, A.; Zhao, Z.; Sundaram, N.; Satish, N.; Sankaran, R.; Jackson, J.; Schwan, K.: Data tiering in heterogeneous memory systems. In: *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys*. 15:1–15:16, 2016.

- [Ga20] Gamrath, G.; Anderson, D.; Bestuzheva, K.; Chen, W.-K.; Eifler, L.; Gasse, M.; Gemander, P.; Gleixner, A.; Gottwald, L.; trin Halbig, K.; Hendel, G.; Hojny, C.; Koch, T.; Bodic, P. L.; Maher, S. J.; Matter, F.; Miltenberger, M.; Mühmer, E.; jamin Müller, B.; Pfetsch, M. E.; Schlösser, F.; Serrano, F.; Shinano, Y.; Tawfik, C.; Vigerske, S.; Wegscheider, F.; Weninger, D.; Witzig, J.: The SCIP Optimization Suite 7.0, [http://www.optimization-online.org/DB\\_HTML/2020/03/7705.html](http://www.optimization-online.org/DB_HTML/2020/03/7705.html), 2020, visited on: 10/04/2022.
- [Go22a] Google, LLC: Google Benchmark: A Microbenchmark Support Library, <https://github.com/google/benchmark>, 2022, visited on: 10/08/2022.
- [Go22b] Google, LLC: Google OR-Tools, <https://developers.google.com/optimization>, 2022, visited on: 10/04/2022.
- [Gu21a] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual, <https://www.gurobi.com/documentation/9.5/refman/index.html>, 2021, visited on: 10/04/2022.
- [Gu21b] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual - Parameter Documentation: MIPGap, <https://www.gurobi.com/documentation/9.5/refman/mipgap2.html#parameter:MIPGap>, 2021, visited on: 10/05/2022.
- [HB22] Hilprecht, B.; Binnig, C.: Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. Proc. VLDB Endow. 15/11, pp. 2361–2374, 2022.
- [He21] Heinzl, L.; Hurdelhey, B.; Boissier, M.; Perscheid, M.; Plattner, H.: Evaluating Lightweight Integer Compression Algorithms in Column-Oriented In-Memory DBMS. In: International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS. Pp. 26–36, 2021.
- [HSY01] Hsu, W. W.; Smith, A. J.; Young, H. C.: I/O reference behavior of production database workloads and the TPC benchmarks - an analysis at the logical level. ACM Trans. Database Syst. 26/1, pp. 96–143, 2001.
- [HWR14] Höppner, B.; Waizy, A.; Rauhe, H.: An Approach for Hybrid-Memory Scaling Columnar In-Memory Databases. In: International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS. Pp. 64–73, 2014.
- [HWW11] Hart, W. E.; Watson, J.-P.; Woodruff, D. L.: Pyomo: modeling and solving mathematical programs in Python. Mathematical Programming Computation 3/3, pp. 219–260, 2011.
- [La22] Lasch, R.; Legler, T.; May, N.; Scheirle, B.; Sattler, K.-U.: Cost Modelling for Optimal Data Placement in Heterogeneous Main Memory. Proc. VLDB Endow. 15/11, pp. 2867–2880, 2022.
- [Lo03] Lougee-Heimer, R.: The Common Optimization Interface for Operations Research: Promoting open-source software in the operations research community. IBM J. Res. Dev. 47/1, pp. 57–66, 2003.

- [Lo19] Lomet, D. B.: Cost/Performance in Modern Data Stores: How Data Caching Systems Succeed. In: Proceedings of the IEEE International Conference on Data Engineering, ICDE. P. 140, 2019.
- [Ma02] Mandelman, J. A.; Dennard, R. H.; Bronner, G. B.; DeBrosse, J. K.; Divakaruni, R.; Li, Y.; Raden, C. J.: Challenges and future directions for the scaling of dynamic random-access memory (DRAM). IBM J. Res. Dev. 46/2-3, pp. 187–222, 2002.
- [Ma16] Ma, L.; Arulraj, J.; Zhao, S.; Pavlo, A.; Dulloor, S. R.; Giardino, M. J.; Parkhurst, J.; Gardner, J. L.; Doshi, K. A.; Zdonik, S. B.: Larger-than-memory data management on modern storage hardware for in-memory OLTP database systems. In: Proceedings of the International Workshop on Data Management on New Hardware, DaMoN. 9:1–9:7, 2016.
- [Ma21] Ma, L.; Zhang, W.; Jiao, J.; Wang, W.; Butrovich, M.; Lim, W. S.; Menon, P.; Pavlo, A.: MB2: Decomposed Behavior Modeling for Self-Driving Database Management Systems. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. Pp. 1248–1261, 2021.
- [ÖTT17] Özcan, F.; Tian, Y.; Tözün, P.: Hybrid Transactional/Analytical Processing: A Survey. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. Pp. 1771–1775, 2017.
- [Pe19] Peng, I. B.; McFadden, M.; Green, E. W.; Iwabuchi, K.; Wu, K.; Li, D.; Pearce, R.; Gokhale, M. B.: UMap: Enabling Application-driven Optimizations for Page Management. In: Workshop on Memory Centric High Performance Computing, MCHPC@SC. Pp. 71–78, 2019.
- [Pl14] Plattner, H.: The Impact of Columnar In-Memory Databases on Enterprise Systems. Proc. VLDB Endow. 7/13, pp. 1722–1729, 2014.
- [RSB22] Richly, K.; Schlosser, R.; Boissier, M.: Budget-Conscious Fine-Grained Configuration Optimization for Spatio-Temporal Applications. Proc. VLDB Endow. 15/13, pp. 4079–4092, 2022.
- [Sh20] Shiratake, S.: Scaling and Performance Challenges of Future DRAM. IEEE International Memory Workshop, IMW/, pp. 1–3, 2020.
- [Va00] Valentin, G.; Zuliani, M.; Zilio, D. C.; Lohman, G. M.; Skelley, A.: DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In: Proceedings of the IEEE International Conference on Data Engineering, ICDE. Pp. 101–110, 2000.
- [Vi21] Viswanathan, V.; Kumar, K.; Willhalm, T.; Lu, P.; Filipiak, B.; Sakthivelu, S.: Intel Memory Latency Checker, <https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html>, 2021, visited on: 10/04/2022.

- [Vo20] Vogel, L.; van Renen, A.; Imamura, S.; Leis, V.; Neumann, T.; Kemper, A.: Mosaic: A Budget-Conscious Storage Engine for Relational Database Systems. *Proc. VLDB Endow.* 13/11, pp. 2662–2675, 2020.
- [We22] Weisgut, M.; Ritter, D.; Boissier, M.; Perscheid, M.: Separated Allocator Metadata in Disaggregated In-Memory Databases: Friend or Foe? In: *IEEE International Parallel and Distributed Processing Symposium, IPDPS Workshops*. Pp. 1202–1208, 2022.
- [Wu21] Wu, K.; Guo, Z.; Hu, G.; Tu, K.; Alagappan, R.; Sen, R.; Park, K.; Arpaci-Dusseau, A. C.; Arpaci-Dusseau, R. H.: The Storage Hierarchy is Not a Hierarchy: Optimizing Caching on Modern Storage Devices with Orthus. In: *19th USENIX Conference on File and Storage Technologies*. Pp. 307–323, 2021.

# Workload-Aware Contention-Management in Indexes for Hierarchical Data

Kevin Wellenzohn,<sup>1</sup> Michael H. Böhlen,<sup>2</sup> Sven Helmer,<sup>3</sup> Marcel Reutegger<sup>4</sup>

**Abstract:** Queries in hierarchical databases (HDBs) often combine predicates referring to values of node properties with path predicates relating to the structure, which are called property-and-path (PP) queries. Usually, PP indexes are used to support these types of queries efficiently. In an environment in which HDBs are updated concurrently, we encounter conflicts which may lead to transaction aborts. We identify *preventable aborts* caused by conflicts in the index, while the operations in the actual database are executed without any problems. These index conflicts are due to the deletion of a path in the index concurrently taking place with an insertion underneath a node on the deleted path. We leverage recent workload information to detect and suspend the deletion of substructures in PP indexes that are likely to conflict with concurrent insertions. However, the suspension of these deletions has a detrimental effect on the query performance, which means this becomes a tradeoff between the number of transaction aborts and the speed of the query evaluation. We implement our approach in Apache Jackrabbit Oak and FOEDUS, experimentally investigate the tradeoff, and show how to balance the effects to maximize the transactional throughput for a given workload.

**Keywords:** hierarchical databases; structural indexes; concurrency control

## 1 Introduction

A lot of the data in business and engineering applications, such as bills of materials [Fi13], enterprise asset hierarchies [Fi13], and business rules [Lo15], is organized in a hierarchical way. Additionally, many NoSQL content stores manage hierarchical data, e.g. in the form of JSON. Similar to relational databases, though, in which we index a subset of attributes in a relation to speed up query evaluation, in hierarchical databases (HDBs), we also often index a subset of nodes in a hierarchy relevant for frequent queries. This set of nodes is application-dependent and we assume that a user flags these nodes, which are then indexed by the system.

Clearly, in a multi-user environment, node indexes can become a bottleneck if nodes are frequently updated concurrently. This leads to conflicts not just on the node level, but may also result in path conflicts on common ancestor nodes of updates. We show how to prevent path conflicts in node indexes that would otherwise lead to transaction aborts. Figure 1a

---

<sup>1</sup> University of Zurich, Dept of Informatics, Binzmühlestrasse 14, 8050 Zurich, Switzerland wellenzohn@ifi.uzh.ch

<sup>2</sup> University of Zurich, Dept of Informatics, Binzmühlestrasse 14, 8050 Zurich, Switzerland boehlen@ifi.uzh.ch

<sup>3</sup> University of Zurich, Dept of Informatics, Binzmühlestrasse 14, 8050 Zurich, Switzerland helmer@ifi.uzh.ch

<sup>4</sup> Adobe Systems, Barfusserplatz 6, 4051 Basel, Switzerland mreutegg@adobe.com

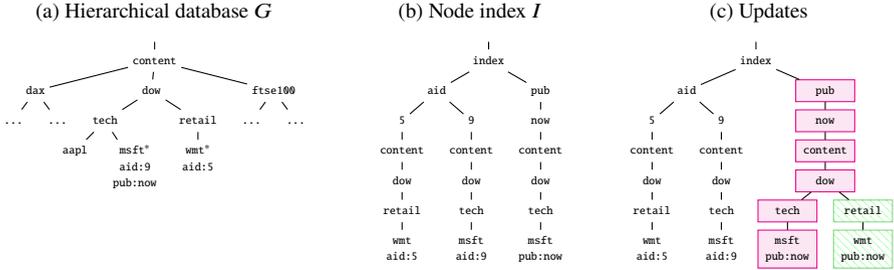


Fig. 1: Transactions  $T_i$  and  $T_j$  conflict when  $T_i$  deletes index node `msft` and its ancestors upwards (red nodes), while  $T_j$  adds a new child to a deleted ancestor (green hatched nodes).

shows an example of a content management system (CMS), such as Adobe Experience Manager [Ad23] or Magnolia [Ni06], built on top of an HDB. In this application scenario, users change webpages in a private workspace of the CMS and, when finished, flag them as being publishable: this adds a property `pub` with the value `now` to publishable nodes. For example, in Figure 1a, the node `msft` is ready to be published. Eventually, the CMS pushes the changes to the webserver and removes the property `pub:now` from the node `msft`. We marked the indexed nodes in the HDB with an asterisk (\*) to make them easier to spot and Figure 1b shows the corresponding index, containing the flagged nodes and their ancestors. We now run the two transactions  $T_i$  and  $T_j$  on this database:  $T_i$  removes the property `pub` from the node `msft`, while  $T_j$  concurrently adds `pub:now` to the node `wmt`. These updates need to be propagated to the index (Figure 1c).  $T_i$ 's deletion of index node `msft` propagates upwards: the empty path `pub/now/content/dow/tech/msft` is deleted (red nodes) since we do not have any nodes with the property `pub` anymore. Concurrently,  $T_j$  needs to add the green nodes `retail/wmt` below `dow` to update the index. This results in a path conflict between  $T_i$  and  $T_j$  since  $T_i$  wants to delete a path, while  $T_j$  wants to insert a branch on this path. The conflict arises at the shared ancestor nodes. In our example, the nodes `msft` and `wmt` in the index have the lowest common ancestor `dow` and share the path from `dow` to the root. We propose a technique identifying problematic regions in node indexes leading to conflicts, i.e., regions in which the same shared ancestor nodes are frequently inserted and deleted. In our example, if we had kept the path from `dow` upwards in the index when removing node `msft`, anticipating an insertion, we could have inserted node `wmt` without any problems. However, this comes at a price: we temporarily keep purposeless nodes in the index, slowing down query evaluation. We experimentally show how to balance reducing contention with query performance to maximize the throughput.

In summary, we make the following contributions:

- We describe and define *preventable aborts*, which are aborts caused by propagated node insertions and deletions in node indexes for hierarchical databases.

- We introduce the notion of *node volatility*, which allows us to identify nodes that are repeatedly involved in preventable aborts.
- We develop the *robust node index* (RNI), which detects and suspends the deletion of volatile nodes to decrease the number of preventable aborts significantly.
- We implemented RNI in Apache Jackrabbit Oak [Ap22] and FOEDUS [Ki15] and evaluated it experimentally with different workloads and datasets. We show that making the node index *robust* is more effective than (a) alternative concurrency-control protocols reducing aborts [WK16] and (b) lazy techniques for node deletions in indexes [Lo04, LS97], increasing the throughput by up to a factor of six.

## 2 Related Work

High-contention workloads are a significant bottleneck for database systems [Ap17, Ha17, RFA16, RTA14, Ti18]. We discuss approaches that deal with contention (a) on the level of the concurrency-control protocol and (b) on the level of the index.

The most similar approach to RNI among the protocol-level approaches is MOCC [WK16] (that is the reason why we chose it for the experimental evaluation). It starts by using optimistic concurrency-control (OCC) to synchronize accesses to records. However, it also monitors the number of aborts caused by a record due to concurrent accesses. If this number reaches a certain threshold, the record is regarded as hot and MOCC switches to a pessimistic locking protocol to reduce the number of aborts. Like MOCC, in RNI we monitor the load on heavily contentious nodes and switch to a different mode when necessary. In the following, we briefly describe other protocol-level approaches. Yuan et al. [Yu16] reduce the number of aborts in OCC by aborting a transaction only if an *essential pattern* exists between transactions, which is more restrictive than the read-write conflict OCC checks for. Similarly, Bumper [DR13] only aborts a transaction if a so-called *triad* (conceptually similar to an essential pattern) is detected. Tian et al. [Ti18] propose a contention-aware locking scheme that reduces the overall lock-waiting times. They choose which transaction  $T$  to grant a lock to based on the number of other transactions that depend on  $T$ 's progress. Johnson et al. [JPA09] reduce contention in the lock manager by passing hot locks directly from transaction to transaction, without releasing and re-acquiring them. QURO [YC16] analyzes program code and reorganizes the code within transactions to reduce contention by acquiring a lock as late as possible. Deterministic concurrency-control has been proposed to reduce synchronization in replicated databases [TA10]. A transaction acquires all locks at its start, which means that transactions competing for exclusive access to a contended record must execute in serial order [Th12]. This prevents conflicts and aborts due to deadlocks at the expense of concurrency (especially under contention). Calvin [Th12] and other deterministic systems require that the read/write sets of transactions be known a priori [Ha17], which is not the case in our application scenario.

Frequently inserting and deleting nodes into and from indexes is a known concurrency bottleneck [LY81, LS97]. Lomet et al. [Lo04, LS97] propose to defer node deletions during updates in B-trees. During deletion, the key is (eagerly) removed from the correct leaf and if it becomes underutilized, the node deletion is deferred and processed later. When to exactly process deferred operations is not specified [LS97], though. Even though there is a lot of work on indexing hierarchical data [HL11, Sh15], concurrency control (CC) specifically for indexes in HDBs has received little attention in comparison to CC for HDBs in general [Be15, Be11, Fi02, HHL06]. For instance, there are path indexes only considering the structure, such as DataGuides [GW97] and APEX [CMS02], and indexes that consider the structure and values, such as IndexFabric [Co01] and CAS (content-and-structure) indexes [Ma15, WBH20]. However, none of these papers discuss concurrency control and we believe there is still untapped potential in this area. For example, node deletions in HDB indexes can be suspended as long as the indexed values are removed during the deletion. In general, this is not possible for CC in the HDB itself, as the actual removal of the node is part of a transaction's semantics. Workload-aware indexing has been shown to improve index query and/or update performance [CMS02, Id11, TYJ09]. APEX [CMS02] optimizes frequently queried paths in XML databases. QU-Trade [TYJ09] uses the recent workload to balance the cost of writing/reading frequently updated/queried objects. Again, none of these approaches discuss concurrency control. In contrast, adaptive indexing incrementally sorts and refines an index during query execution [Id11], sketching ideas on how to realize CC in the future work section. This promise is delivered in [Gr14], which provides more details on concurrency control. Queries can cause contention if they concurrently attempt to optimize overlapping query-ranges. In that case adaptive indexing forgoes the chance to optimize the index and skips the optional optimization.

### 3 Background

#### 3.1 Data Model

We model a database  $G$  as an unordered tree that is defined as a set of nodes  $G = \{n_1, n_2, \dots\}$ .<sup>5</sup> A node  $n = / \lambda_1 / \dots / \lambda_x$  is uniquely identified by the *node labels*  $\lambda_i$  on the path from the root node to node  $n$ . The last node label in this sequence,  $\lambda_x$ , is  $n$ 's label. The label of the root node is the empty string.

**Example 1** Consider the database  $G$  in Fig. 1a. Node  $n = /content/dow/tech/msft$  has label *msft*. If no ambiguity arises, we shorten node labels by using only the initials, hence  $n = /c/d/t/m$ .  $G$  consists of the labels of all its nodes  $\{/, /c, /c/d, /c/d/t, /c/d/r, /c/d/t/a, /c/d/t/m, /c/d/r/w, \dots\}$ . From now on, we denote a node by its label  $\lambda$ ; the full ID can be derived from its ancestors' labels.  $\square$

<sup>5</sup> Based on Apache Jackrabbit Oak's data model [Ap22].

A node may have an arbitrary number of properties. We define the *property set*  $P(G)$  of tree  $G$  as a set of triples  $(n, k, v)$ , which denote that node  $n \in G$  has property  $k$  set to value  $v \neq \epsilon$ . We use the notation  $n[k]_{(G)} = v$  iff  $(n, k, v) \in P(G)$ , and  $n[k]_{(G)} = \epsilon$  iff  $\nexists v((n, k, v) \in P(G))$  to denote that node  $n$  does not have property  $k$  in  $G$ . If it is clear from the context which tree we are referring to, we omit the subscript and write  $n[k] = v$  or  $n[k] = \epsilon$ . A node  $n$  is an ancestor of node  $m$  (and  $m$  is a descendant of  $n$ ) iff  $n = / \lambda_1 / \dots / \lambda_x$  is a prefix of  $m = / \lambda_1 / \dots / \lambda_x / \dots / \lambda_y$  or, stated shortly,  $\text{prefix}(n, m)$ . A node is an ancestor and descendant of itself, i.e.,  $\text{prefix}(n, n)$  is true for every node  $n$ .<sup>6</sup>

**Example 2** Consider Fig. 1 and let  $n = /c/d/t/m$ . We have  $n[\text{pub}]_{(G)} = \text{now}$  before running transaction  $T_i$  and  $n[\text{pub}]_{(G)} = \epsilon$  after running  $T_i$ . Node  $/c/d$  is an ancestor of  $n$ , since  $\text{prefix}(/c/d, /c/d/t/m)$  is true.  $\square$

Typically, queries in HDBs are property-and-path (PP) queries, meaning we need to provide a property, a value to compare to, and a path. As we only consider paths (and not twigs), the order of the siblings in a tree does not matter.

**Definition 1** (PP Query) A PP query  $Q = (k, v, m)$  returns the set of nodes with property  $k$  equal to value  $v$  that are descendants of  $m$ , i.e.,  $\{d \mid d[k] = v \wedge \text{prefix}(m, d)\}$ .  $\square$

### 3.2 The Property-and-Path (PP) Index

A property-and-path (PP) index  $I$  is used to efficiently query all nodes in a subtree that have a property  $k$  set to a value  $v$ . Essentially, a PP index is modeled as an unordered tree, similar to an HDB as described in Section 3.1. The first label of every path in  $I$  is called *index*, the second is the name of a property  $k$ , and the third is a value  $v$  for  $k$ . This is then followed by paths to all nodes in the indexed database  $G$  that have a property  $k$  with a value  $v$  (cf. Figure 1b). In a typical application, a node can have many properties (e.g., author ID *aid* and other metadata), but usually only some are indexed.

**Querying:** Evaluating PP query  $Q = (k, v, / \lambda_1 / \dots / \lambda_x)$  with index  $I$  translates to navigating down the path  $/ \text{index} / k / v / \lambda_1 / \dots / \lambda_x$ , traversing all descendants of  $\lambda_x$ , searching for index nodes  $n$  with  $n[k] = v$ , and returning their corresponding content nodes. These are obtained by truncating the three leading node labels of  $n$ . For example, for index node  $/i/p/n/c/d/t/m$  the corresponding content node is  $/c/d/t/m$ .

**Example 3** Assume we want to find pages under *now* that are ready for publication, i.e., we run the query  $Q = (\text{pub}, \text{now}, /c/d)$  on  $G$ . Using the index  $I$  in Figure 1b, we descend to

<sup>6</sup> This is similar to the ancestor-or-self and descendant-or-self axes in XPath.

node  $/i/p/n/c/d$  and check if any descendant contains the property-value pair  $pub:now$ . This is the case for node  $m = /i/p/n/c/d/t/m$ , thus the query returns  $\{/c/d/t/m\}$ . (As we will see later, there may be unproductive nodes in the index missing the property-value pair  $pub:now$ . These nodes are currently not active and will not be returned.)  $\square$

**Insertion:** An insertion into index  $I$  is described by a triplet  $(k, v, n = /λ_1/.../λ_x)$ , where  $k$  is a property,  $v$  is a value, and  $m$  is a node (that now has a property  $k$  set to value  $v$ ). The insertion is executed as follows. First, the system traverses the nodes along path  $n = /index/k/v/λ_1/.../λ_x$  or creates them if they do not exist yet. Then, the system sets  $n[k] = v$ .

**Example 4** When transaction  $T_j$  adds property  $pub:now$  to node  $/c/d/r/w$  in  $G$  (cf. Figure 1a), we add a branch  $/r/w$  underneath  $/i/p/n/c/d$  in index  $I$  in Figure 1b, setting the property  $pub$  in node  $wmt$  to  $now$ .  $\square$

**Deletion:** A deletion is also described by a triplet  $(k, v, n = /λ_1/.../λ_x)$ . During the deletion, we first descend to node  $n = /index/k/v/λ_1/.../λ_x$  and remove property  $k$  by setting  $n[k] = \epsilon$ . However, it does not stop there. We prune  $n$  and all its ancestors one by one as long as they are a leaf and do not have the property  $k$  (we do not prune the index definition  $/index$ ).

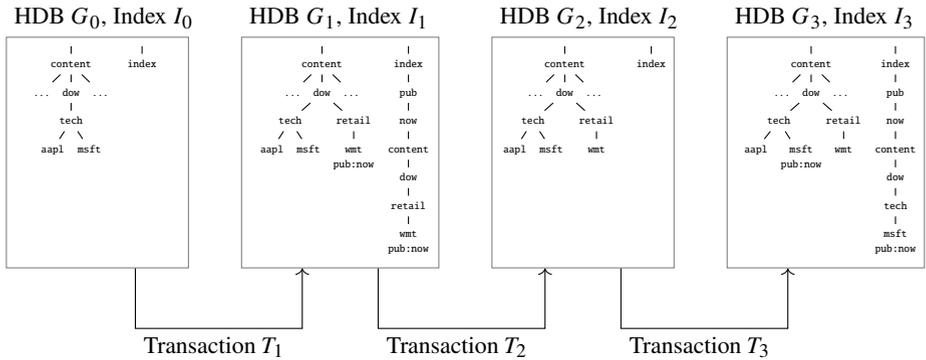
**Example 5** When transaction  $T_i$  removes property  $pub$  from node  $/c/d/t/m$  in  $G$ , the property  $pub$  is removed from index node  $n = /i/p/n/c/d/t/m$ . As  $n$  is now a leaf node without a property, it is deleted. The pruning continues up to  $index$ , essentially removing the path  $p/n/c/d/t/m$  from index  $I$  (cf. Figure 1b).  $\square$

## 4 Conflicts and Aborts

This section describes concurrent operations that lead to conflicts in HDBs. We assume multi-version concurrency control (MVCC) with snapshot isolation. MVCC resolves conflicts by aborting transactions.

### 4.1 Snapshots

The state of an HDB logically progresses from one snapshot of the database to the next as transactions commit. A *history* is a sequence  $H = \langle \dots, G_i \rangle$  of databases (including any indexes) ordered by commit time. A committed HDB  $G_i \in H$  is an immutable *snapshot*. A new transaction  $T_j$  logically creates a mutable copy  $G_j$  of the last committed snapshot



Each transaction's changes to the content subtree:

- ▷ Transaction  $T_1$ : An author adds a webpage wmt with property pub set to now
- ▷ Transaction  $T_2$ : The CMS removes property pub from wmt after pushing it to the webserver
- ▷ Transaction  $T_3$ : An author publishes webpage msft by setting its property pub to now

Fig. 2: A typical CMS-workload in which authors repeatedly publish webpages.

$G_i \in H$  with snapshot  $G_i$  being the *base snapshot* of  $T_j$ . Transaction  $T_j$  applies all its read and write operations on  $G_j$ .

**Example 6** Our running example (see Figure 2) shows an initial HDB  $G_0$  with a corresponding (empty) index  $I_0$ .<sup>7</sup> After running the transactions  $T_1$  to  $T_3$ , one after the other, we have the history  $H$  with the committed snapshots  $G_0$  to  $G_3$ :  $H = \langle G_0, G_1, G_2, G_3 \rangle$ .  $\square$

## 4.2 Conflict Detection and Handling

Before going into the details of resolving conflicts between concurrent transactions, we define basic notions of transactions in HDBs. A transaction  $T_j$  can change a database with two primitives: node-write operations  $wn(n)$ , to insert or delete nodes, and property-write operations  $wp(n, k)$ , to add, delete, or change a property  $k$  of node  $n$ .

**Definition 2** (Write Set) The write set  $\Delta T_j$  of a transaction  $T_j$  is the set of node- and property-write operations in tree  $G_j$ . Let  $G_i$  be  $T_j$ 's base snapshot.  $\Delta T_j$  contains:

1. Node-write operations  $wn(n)$ :

$$wn(n) \in \Delta T_j \Leftrightarrow (n \in G_i - G_j) \vee (n \in G_j - G_i)$$

<sup>7</sup> For the sake of simplicity, we dropped the property aid.

2. *Property-write operations*  $\text{wp}(n, k)$ :

$$\begin{aligned} \text{wp}(n, k) \in \Delta T_j \Leftrightarrow & (n \in G_j - G_i \wedge n[k]_{(G_j)} \neq \epsilon) \vee \\ & (n \in G_i - G_j \wedge n[k]_{(G_i)} \neq \epsilon) \vee \\ & (n \in G_i \cap G_j \wedge n[k]_{(G_i)} \neq n[k]_{(G_j)}) \quad \square \end{aligned}$$

**Example 7** The write set  $\Delta T_1$  of transaction  $T_1$  in Figure 2 contains the following operations:  $\text{wn}(/c/d/r$  and  $/c/d/r/w)$ , creating the node `retail` and then the node `wmt`, and  $\text{wp}(/c/d/r/w, \text{pub}:\text{now})$ , adding the property `pub` with the value `now` to the node `wmt`. Moreover, it also includes the operations  $\text{wn}(/i/p)$ ,  $\text{wn}(/i/p/n)$ ,  $\text{wn}(/i/p/n/c)$ ,  $\text{wn}(/i/p/n/c/d)$ ,  $\text{wn}(/i/p/n/c/d/r)$ ,  $\text{wn}(/i/p/n/c/d/r/w)$ , and  $\text{wp}(/i/p/n/c/d/r/w, \text{pub}:\text{now})$ , updating the index.  $\square$

We have to distinguish different types of conflicts between two concurrent transactions  $T_i$  and  $T_j$ : *path conflicts* and *property conflicts*. Path conflicts include at least one  $\text{wn}$  operation that inserts or deletes a node and are denoted by  $\text{wn-wn}$ ,  $\text{wn-wp}$ , and  $\text{wp-wn}$ . We encounter a  $\text{wn-wn}$  conflict if one transaction adds/deletes a node while the other adds/deletes one of its descendants, i.e., the label of one node is a prefix of the other. A  $\text{wn-wp}$  or  $\text{wp-wn}$  conflict exists if one transaction deletes a node, while the other adds, changes, or deletes any property on the *same* node. Property conflicts ( $\text{wp-wp}$  conflicts) occur when  $T_i$  and  $T_j$  simultaneously try to change the *same* property on the *same* node.

**Definition 3** (Path Conflict) We have a *path conflict* between concurrent transactions  $T_i$  and  $T_j$  iff at least one of the following conflicts occurred:

1. *wn-wp conflict*:  $\exists n, k (\text{wn}(n) \in \Delta T_i \wedge \text{wp}(n, k) \in \Delta T_j)$
2. *wp-wn conflict*:  $\exists n, k (\text{wp}(n, k) \in \Delta T_i \wedge \text{wn}(n) \in \Delta T_j)$
3. *wn-wn conflict*:  $\exists n, m (\text{wn}(n) \in \Delta T_i \wedge \text{wn}(m) \in \Delta T_j \wedge (\text{prefix}(n, m) \vee \text{prefix}(m, n))) \square$

**Definition 4** (Property Conflict) A *property conflict*, i.e., *wp-wp conflict*, exists between concurrent transactions  $T_i$  and  $T_j$  iff  $\exists n, k (\text{wp}(n, k) \in \Delta T_i \wedge \text{wp}(n, k) \in \Delta T_j) \square$

**Example 8** Assume that transactions  $T_4$  and  $T_5$  start concurrently in HDB  $G_3$  (see Figure 3), hence  $G_3$  becomes  $T_4$ 's and  $T_5$ 's base snapshot.  $T_4$  and  $T_5$  run into a path conflict ( $\text{wn-wn}$ ), since the former deletes a node under which the latter adds a child. The conflicting operations in the write sets of  $T_4$  and  $T_5$ ,  $\text{wn}(/i/p/n/c/d) \in \Delta T_4$  and  $\text{wn}(/i/p/n/c/d/r) \in \Delta T_5$ , are highlighted in red in Figure 3.  $\square$

When a transaction  $T_j$  attempts to commit, a verification phase checks whether  $T_j$  conflicts with a concurrent transaction. If a conflict is detected one of the involved transactions

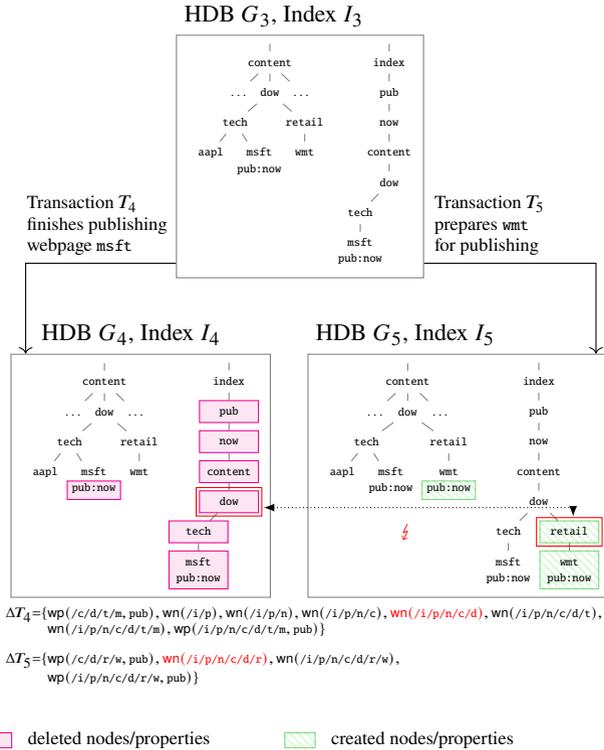


Fig. 3: Transactions  $T_4$  and  $T_5$  conflict, because  $T_4$  deletes index node dow, while  $T_5$  adds child retail.

has to abort. Oak implements the first-committer-wins rule [Be95], which means that the transaction that issues the commit first is allowed to commit, while the other is aborted (other policies, such as timestamp-based priority to favor older transactions are also possible). In our running example  $T_4$  commits first and therefore  $T_5$  must abort due to the conflict shown above.

Clearly, if there is a conflict caused by operations in the database, one transaction has to abort. Two different transactions concurrently changing the same node at the same time are just not compatible. However, what is particularly interesting in Example 8 is that the conflict is caused by operations updating the index. The operations updating the actual database  $G_3$  are perfectly fine, as they update properties in two completely different nodes. It turns out that if a conflict occurs only in the index, we sometimes have options to avoid such an abort. We take a closer look at this in the following section.

## 5 The Robust Node Index (RNI)

## 5.1 Volatile Nodes

Path conflicts occur frequently in index hotspots where transactions insert and delete nodes sharing a large number of ancestors. We call nodes that are repeatedly inserted and deleted *volatile*. These are a main source for path conflicts in indexes. We propose the *robust node index (RNI)* that detects and manages volatile index nodes. RNI suspends the deletion of a volatile index node, as we expect the node to be inserted again soon. Not repeatedly deleting and inserting a volatile node  $n$  means that node-write operations on  $n$ ,  $\text{wn}(n)$ , are avoided, reducing contention and, consequently, the number of aborting transactions.

We define the volatility of a node  $n$  as the number of times  $n$  was inserted or deleted. This corresponds to checking the number of  $\text{wn}(n)$  operations that have been executed (cf. Definition 2). In order to do so, we look at the recent transactional workload, which is defined by a sliding window  $\text{SW}(H, L)$  of length  $L$  over history  $H$ .  $\text{SW}(H, L)$  denotes the set of transactions that committed over the last  $L \geq 0$  time units. Let  $t_{\text{now}}$  be the current time and  $t(T)$  be the commit time of transaction  $T$ , then  $\text{SW}(H, L) = \{T_j \mid t(T_j) \in (t_{\text{now}} - L, t_{\text{now}}] \wedge G_j \in H\}$ .

**Definition 5 (Volatile Node)** *A node  $n$  is volatile in history  $H$  iff the number of transactions in sliding window  $\text{SW}(H, L)$  that executed a  $\text{wn}(n)$  operation is at least equal to the volatility threshold  $\tau$ , i.e.,*

$$|\{T \mid T \in \text{SW}(H, L) \wedge \text{wn}(n) \in \Delta T\}| \geq \tau \quad \square$$

**Example 9** *Consider index node  $n = /i/p/n/c/d$  in index  $I_3$  in Figure 2. Assuming time  $t_{\text{now}} = 11$  and sliding window length  $L = 10$ , Figure 4 shows the commit times  $t(T_1)$ ,  $t(T_2)$ , and  $t(T_3)$  of the transactions we ran on our HDB. Since all commit times lie in the sliding window,  $\text{SW}(H_1, L) = \{T_1, T_2, T_3\}$ . All these transactions either insert or delete  $n$ , thus  $\forall T \in \text{SW}(H_1, L) : \text{wn}(n) \in \Delta T$  and for a volatility threshold  $\tau \leq 3$ , node  $n$  is volatile.  $\square$*

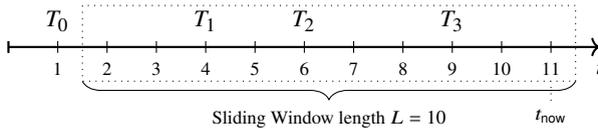


Fig. 4: Transactions  $T_1$  through  $T_3$  from Figure 2 are contained in the sliding window  $\text{SW}(H_1, 10)$ .

RNI checks for volatile nodes during the pruning of nodes in the index. The deletion of pair  $(v, m)$  with path  $m = /l_1/\dots/l_x$  and value  $v$  from index  $I$  is described in Algorithm 1. RNI descends to index node  $n = /index/k/v/l_1/\dots/l_x$ , deletes  $n$ 's property  $k$ , and tries to prune  $n$ . RNI only prunes a node if three conditions are satisfied: (a) it is (or has become) a leaf, (b) it does not have property  $k$ , and (c) it is *not* volatile.<sup>8</sup>

<sup>8</sup> It also does not prune the topmost node, *index*.

---

**Algorithm 1: Deletion in RNI**


---

**Input:** Index  $I$ , pair  $(v, m)$ , and history  $H$ .  $k$  is a property,  $v$  a value, and  $m = /λ_1/λ_2/.../λ_x$  a node.

```

1  $n \leftarrow /index/k/v/λ_1/λ_2/.../λ_x$ 
2  $n[k] \leftarrow \epsilon$ 
3 while  $n \neq /index \wedge isLeaf(n) \wedge n[k] = \epsilon \wedge \neg volatile(n)$  do
4    $u \leftarrow n$ 
5    $n \leftarrow parent\ of\ n$ 
6   Delete node  $u$ 
    
```

---

**Example 10** *RNI prunes node  $u_1 = /i/p/n/c/d/t/m$  in response to transaction  $T_4$  deleting property `pub` from node `/c/d/t/m`. The node can be deleted since  $u_1$  is not volatile. The pruning propagates to  $u_1$ 's parent node  $u_2 = /i/p/n/c/d/t$ , which can also be pruned. However, the parent of  $u_2$ ,  $u_3 = /i/p/n/c/d$ , is not pruned and the deletion is not propagated farther up the index, because  $u_3$  is volatile (cf. Example 9). Since `/i/p/n/c/d` is no longer deleted by  $T_4$ ,  $T_5$ 's insertion of a child node is not a conflict anymore.  $\square$*

Currently, we have implemented the tracking of volatile nodes in a naive fashion, i.e., we just count the number of insertions and deletions executed on each node. However, the performance of volatility tracking can be improved considerably by employing algorithms from stream processing for finding frequent items. For our purposes, we do not need exact numbers, so an approximation is enough, which improves the performance even more. For instance, Cormode and Hadjieleftheriou use a sketch algorithm for finding frequent items in data streams [CH10].

## 5.2 Preventable Aborts

As we have seen in Example 10, we can avoid aborting a transaction when a path conflict occurs in the index by not deleting volatile nodes. We now take a closer look at these *preventable aborts*.

**Definition 6** (Preventable Abort) *Let  $T_j$  be a transaction that is aborted due to a conflict with transaction  $T_i$ .  $T_j$ 's abort is preventable iff each conflict with  $T_i$  is a path conflict in the index.  $\square$*

**Lemma 1** *Let  $T_i$  and  $T_j$  be two concurrent transactions.  $T_i$ 's and  $T_j$ 's write operations on an existing node  $n$  in a RNI index cannot cause a preventable abort if  $n$  is volatile or has a volatile descendant.  $\square$*

**Proof**  $T_i$  and  $T_j$  can only cause a path conflict if both  $T_i$  contain operations changing the same property  $k$ . If they change different properties, the index updates take place in completely

separate branches of  $I$ . Let node  $n$  be a node in RNI  $I$ . Let  $d$  be a volatile descendant of  $n$  (recall that  $n$  is a descendant of itself). Since  $d$  is volatile, neither  $T_i$  nor  $T_j$  can prune  $d$  or any of its ancestors, including  $n$ . Therefore  $\text{wn}(n) \notin \Delta T_i$  and  $\text{wn}(n) \notin \Delta T_j$ . As a consequence, we can rule out any path conflict (i.e.,  $\text{wn-wn}$ ,  $\text{wn-wp}$ , and  $\text{wp-wn}$  conflicts). The only possible conflict between  $T_i$  and  $T_j$  is a  $\text{wp-wp}$  property conflict on property  $k$ . However, this is a property conflict, i.e.,  $\text{wp}(n, k) \in \Delta T_i$  and  $\text{wp}(n, k) \in \Delta T_j$ , for which an abort is *not* preventable.  $\square$

### 5.3 Unproductive Nodes

While not deleting volatile nodes reduces the number of aborting transactions, this slows down query evaluation, thus it is a trade-off. We call non-deleted volatile nodes *unproductive*, as they have to be traversed during query evaluation, but do not contribute to the result set of the query. A characteristic of an unproductive node in an RNI is that neither the node itself nor any of its descendants have a value for a property.

**Definition 7** (Unproductive Node) *An index node  $n$  is unproductive in tree  $G$  iff no descendant of  $n$  has any property:*

$$\forall d((d \in G \wedge \text{prefix}(n, d)) \Rightarrow \nexists k(d[k] \neq \epsilon)) \quad \square$$

**Example 11** *After running  $T_4$  in Example 10, the index node  $/i/p/n/c/d$  and its ancestors are unproductive because they do not have any properties. Nevertheless, during query evaluation, we still traverse these nodes.*  $\square$

### 5.4 Parameterization

**Volatility Threshold  $\tau$**  Let us consider two extreme values for  $\tau$ . With  $\tau = \infty$ , RNI is identical to a basic PP index as described in Section 3.2, since a node never becomes volatile. With  $\tau = 0$ , index nodes are never pruned and the performance of queries deteriorates, because many nodes will be unproductive. Additionally, the index will keep growing, meaning we also waste a lot of space. Our goal is to choose threshold  $\tau$  that best balances the number of path conflicts and query runtime to maximize the throughput.

This tradeoff is workload-dependent; a write-heavy workload calls for small values of  $\tau$  to reduce the number of aborts, while a read-heavy workload benefits from larger values of  $\tau$  so that query performance does not suffer too much. In a balanced workload, moderate values of  $\tau$  are most promising. Nodes in mostly static subtrees with few updates and few conflicts, which constitute the largest part of the index, are pruned and queries perform well. Nodes in dynamic subtrees that are repeatedly inserted and deleted are already retained after a small number of updates, minimizing the number of aborts. We investigate this tradeoff in our experimental evaluation (Section 6).

**Sliding Window Length  $L$**  Parameter  $L$  determines how much of the recent workload is used to determine whether a node is volatile. If we set  $L = 0$ , the sliding window is empty and a node cannot become volatile unless  $\tau = 0$ . As we increase  $L$ , a node is more likely to be classified as volatile, because we consider a larger portion of history  $H$ . The sliding window length  $L$  is naturally upper-bounded by the time frame that history  $H$  covers. For instance, Oak periodically runs a garbage collection to delete old snapshots in  $H$  and snapshots are retained for a minimum amount of time (the default is 24 hours). We choose  $L = 24$  hours to use all workload information that Oak provides.

## 6 Experimental Evaluation

Our experimental evaluation considers synthetic and real-world datasets (see Section 6.1) as well as different workloads (see Section 6.2), i.e., read-heavy and write-heavy scenarios. We organize the evaluation as follows:

1. In Section 6.3 we show how to calibrate RNI. We experimentally determine the optimal threshold  $\tau$  that balances query performance and number of aborts. We also look at the impact of the length of the sliding window.
2. In Section 6.4, we compare RNI to an enhanced basic PP index running the concurrency-control protocol MOCC [WK16] that was specifically designed to reduce the number of aborts. We demonstrate that a basic PP index modified with MOCC still suffers from many path conflicts and show that RNI provides a better throughput.
3. Finally, in Section 6.5, we investigate an approach deferring node deletions to improve concurrency during updates (proposed by Lomet et al. [Lo04, LS97]). However, this only delays the conflicts: the deferred deletions often clash with regular user transactions later on and RNI's performance is still better.

### 6.1 Preliminaries

We use real-world and synthetic datasets in our experimental evaluation. The real-world dataset is the Dell website<sup>9</sup> and contains 12,244,893 nodes. A node has an average (maximum) depth of 13.68 (24) and an average (maximum) fanout of 2.88 (1729). The synthetic dataset is a binary tree of depth 19 and contains  $2^{20} - 1 \approx 1\text{M}$  nodes. Using a binary tree increases the likelihood of path conflicts, so this dataset simulates a kind of worst-case scenario.

<sup>9</sup> <https://dell.com>; Dell uses AEM [Ad23] as CMS and Oak as HDB for its website. The Dell dataset has been extracted from a dump of Oak.

Each experiment was run for five minutes. At the beginning of an experiment, the index is pre-populated with 10% of the nodes from the dataset. The experiments are conducted on virtual machines, each having 8 CPU cores and 32GB of RAM. Unless stated otherwise, we use 8 threads that run concurrent transactions.

## 6.2 Workloads

We use two types of transactions, *writers* and *readers*, that simulate the publishing of webpages. Each writer picks a set of 50 content nodes, adds a property, and updates the index accordingly. A subsequent writer removes this property from the same content nodes and updates the index. A reader simulates the background process that looks for publishable webpages by executing a PP query against the index. We use three variations of this workload that differ in the ratio between writer and reader transactions (cf. Table 1).

| Workload        | Abbrev. | Writer:Reader Ratio |
|-----------------|---------|---------------------|
| Write-Intensive | WI      | 5:1                 |
| Balanced        | BA      | 1:1                 |
| Read-Intensive  | RI      | 1:5                 |

Tab. 1: The three considered workloads.

**Writers** We generate the writer transactions to provoke preventable aborts. We do so by splitting the database  $G$  into the same number of partitions as concurrent transactions (or threads) and assigning them to writers. Writers only randomly modify properties of nodes in their partition, i.e., there are no conflicts in the database itself, we can only have path conflicts in the index. The partitioning is done as follows. We assign each node  $n$  a unique rank  $r$ ,  $1 \leq r \leq N$ , with  $N$  being the number of nodes in the tree  $G$ . The rank of each node is determined by an inverse level-order traversal of  $G$ , i.e., the first leaf has rank 1 and the root has rank  $N$ . A node with rank  $r$  belongs to partition  $p = r \bmod P$ , where  $P$  is the number of partitions, thus each partition contains  $\lfloor N/P \rfloor$  nodes. When determining the write set of a write transaction, the  $j$ -th node in a partition is picked with a probability of  $\text{Zipf}(j, \lfloor N/P \rfloor, s_w)$ , where  $s_w$  is the skew (of the write transactions). The Zipfian distribution  $\text{Zipf}(j, J, s)$  is equal to  $(j^s \sum_{i=1}^J \frac{1}{i^s})^{-1}$ , where  $J$  is the number of elements,  $j$  the position of an element ( $1 \leq j \leq J$ ), and  $s$  the skew ( $s = 0$  being the uniform distribution). The default value of  $s_w$  in our experiments is 1.

**Readers** A reader executes a single PP query  $Q = (k, v, /\lambda_1/ \dots / \lambda_d)$ . The root of the traversed subtree is randomly chosen among all nodes at a certain depth  $d$  (in our experiments we choose  $d = 8$ ). For our synthetic dataset, which is a binary tree with depth 19, a PP query with  $d = 8$  traverses a subtree with at most  $2^{19-8+1} = 4096$  nodes. Let  $N_d$  be the number of nodes at depth  $d$ . The  $j$ -th node among all nodes at depth  $d$  is picked with

probability  $Zipf(j, N_d, s_r)$ , where parameter  $s_r$  is the reader skew. The default value of  $s_r$  in our experiments is 1.

### 6.3 Calibration of RNI and Comparison with Basic PP Index

We begin with the calibration of the threshold  $\tau$  and show how it affects the tradeoff between contention (expressed as the number of preventable aborts) and the query performance (expressed as the number of nodes read). The results of the experimental evaluation in Figure 5 also serve as a comparison of RNI with a basic PP index. The measurements at the far right-hand side of every diagram ( $\tau = 1000$ ) represent the performance of a basic PP index: all the curves flatten off at that point and continue on the same level for even larger values of  $\tau$ .

#### 6.3.1 Volatility Threshold $\tau$

**Volatile Nodes** The first column of diagrams in Figure 5 shows the percentage of index nodes that are volatile, depending on the threshold  $\tau$ , at the end of an experimental run. Clearly, the smaller  $\tau$ , the more volatile nodes there are. For  $\tau = 1$ , between 40% and 70% of all index nodes are volatile for the synthetic dataset, while the numbers are lower for the Dell dataset at around 10% to 20% (this dataset is larger and, thus, each individual node is inserted and deleted less frequently). Also, write intensive workloads have more volatile nodes than read-intensive ones, as they contain more insert and delete operation. With increasing  $\tau$  the percentage of volatile nodes eventually reaches zero, which is equivalent to a basic PP index: it has no volatile nodes.

**Abort Ratio** The second column of diagrams in Figure 5 illustrates the impact of  $\tau$  on the abort ratio of transactions. For small values of  $\tau$ , we can eliminate almost all preventable aborts, as many nodes become volatile and path conflicts occur rarely. With increasing  $\tau$ , the abort ratio increases. For write-intensive workloads, the abort ratio reaches 50%, while for read-intensive workloads, this ratio is much lower, at about 10%, since read transactions do not conflict with each other. In summary, this confirms that RNI is able to detect and retain index nodes that are responsible for preventable aborts, which are detrimental to the performance of a basic PP index.

**Number of Read Nodes** The third column of diagrams in Figure 5 shows the flip side: while a small value of  $\tau$  reduces the number of aborted transactions, the query performance suffers, as many unproductive volatile nodes have to be traversed. For  $\tau = 1$ , a PP query visits two-and-a-half to four times as many nodes during query evaluation for the synthetic dataset compared to the number of nodes for a large value of  $\tau$ . Due to the larger size of the

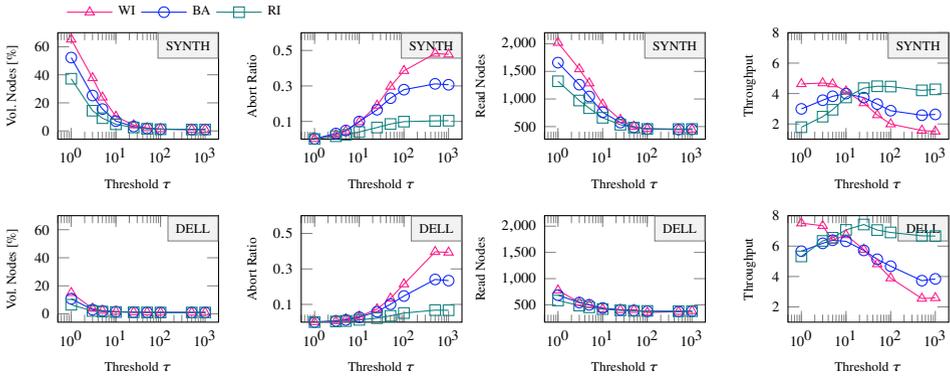


Fig. 5: Threshold  $\tau$  trades query performance and abort ratio to increase the throughput.

Dell dataset and (therefore) fewer volatile nodes, these numbers are smaller. For increasing  $\tau$ , the number of read nodes eventually levels off at just under 500, which is the number of nodes that have to be accessed to answer a query in a basic PP index.

**Throughput** We report normalized values here to make the results comparable to those in Sections 6.4 and 6.5. Normalizing means calculating the ratio between the serial execution of the basic PP index (as a baseline) and the concurrent execution of RNI. The last column of diagrams in Figure 5 shows the results for our experiments on throughput. As already mentioned, RNI trades the reduction of contention against query performance. However, the situation is not quite that simple. For write-intensive workloads, aborts are the major performance bottleneck as opposed to query performance, so  $\tau = 1$  yields the best throughput (recall that we run the experiments on an eight-core machine, so a throughput of eight means perfect parallelization). We observe the most pronounced effect for balanced workloads: here values of around 10 for  $\tau$  feature the best performance, with smaller and larger values showing significantly less performance. For read-intensive workloads, the optimal throughput performance is not as distinctive as for balanced workloads. Moreover, the optimal value for  $\tau$  is shifted to the right, as query performance plays a more important role. Nevertheless, by using an appropriate value of  $\tau$ , we can always achieve a better performance with RNI compared to a basic PP index.

### 6.3.2 Sliding Window Length $L$

Finally, we look at the impact of the length  $L$  of the sliding window.  $L = 0$  mirrors the case  $\tau = \infty$ : in both cases no node can become volatile, so a number greater than zero has to be chosen to see any kind of effect. As we see in Figure 6, the measured numbers for the throughput stabilize quickly.

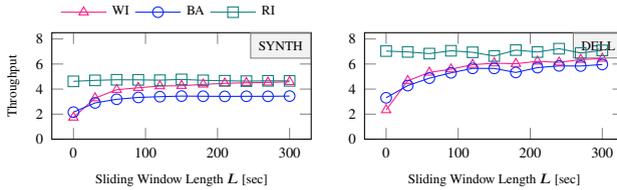


Fig. 6: RPP’s throughput is insensitive to  $L$  as long as  $L > 0$ .

#### 6.4 Comparison with MOCC

Next, we investigate whether a basic PP index running the MOCC protocol [WK16], which we call  $\text{MOCC}_{\text{PP}}$ , is able to compete with our approach RNI. We ran the  $\text{MOCC}_{\text{PP}}$  experiments in the in-memory system FOEDUS [Ki15], in which MOCC is natively implemented. Table 2 illustrates the differences in terms of (normalized) throughput between  $\text{MOCC}_{\text{PP}}$  and RNI (the best result per dataset and approach is shown in boldface). For RNI, we show two rows with results. The first row (optimized) uses the optimal value of  $\tau$  for each of the different workloads. In practice, it will be difficult to tune RNI for every individual workload, so the second row ( $\tau = 10$ ) shows the results for a configuration employing a common value of  $\tau = 10$  for all the workloads.

| workload<br>approach      | SYNTH       |      |             | DELL        |      |             |
|---------------------------|-------------|------|-------------|-------------|------|-------------|
|                           | WI          | BA   | RI          | WI          | BA   | RI          |
| $\text{MOCC}_{\text{PP}}$ | 1.48        | 3.45 | <b>5.94</b> | 1.63        | 3.30 | <b>5.80</b> |
| RNI (optimized)           | <b>4.69</b> | 4.00 | 4.48        | <b>7.51</b> | 6.39 | 7.42        |
| RNI ( $\tau = 10$ )       | <b>4.15</b> | 4.00 | 3.74        | 6.73        | 6.31 | <b>7.07</b> |

Tab. 2: Comparison of normalized throughput between  $\text{MOCC}_{\text{PP}}$  and RNI.

We make a couple of observations here. The higher the ratio of read transactions, the better  $\text{MOCC}_{\text{PP}}$  performs. This does not come as a surprise: for highly contentious workloads, MOCC runs an optimistic concurrency protocol with a low overhead, i.e., no locks are used, and during a validation phase transactions that are in conflict with other transactions have to abort. In read-intensive workloads, conflicts rarely occur. However, when faced with heavy contention, MOCC switches to a pessimistic lock-based protocol to avoid a large number of transactions to abort during the validation phase. While this does bring down the number of aborted transactions, it introduces an overhead in the form of lock management and in a write-intensive workload with many conflicts, transactions have to wait for the release of locks. In case of a deadlock, we may even have to abort transactions. The performance of RNI is much more balanced across the different workloads: it can handle environments with a lot of write conflicts much better than  $\text{MOCC}_{\text{PP}}$ . In summary, there is only one scenario, the read-intensive synthetic workload, for which  $\text{MOCC}_{\text{PP}}$  performs better than RNI. In all other cases, RNI outperforms  $\text{MOCC}_{\text{PP}}$ .

### 6.4.1 MOCC<sub>RNI</sub>: Combining MOCC with RNI

Since RNI and MOCC use orthogonal principles, we can combine the two to obtain an even better approach by running MOCC with volatile nodes. We call this protocol MOCC<sub>RNI</sub>. Figure 7 shows the results for calibrating the parameter  $\tau$  for MOCC<sub>RNI</sub>. Comparing these results to the last two diagrams in the bottom row of Figure 5 (illustrating the tuning of RNI), we see that the performance of MOCC<sub>RNI</sub> is worse than that of RNI for write-intensive and balanced workloads (i.e., workloads with a higher proportion of write transactions). In these cases, the advantage of using a small value of  $\tau$  is offset by the overhead of using a pessimistic locking protocol. Consequently, we should never use small values of  $\tau$  for MOCC<sub>RNI</sub>.

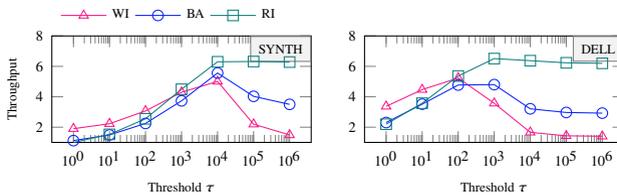


Fig. 7: Optimal thresholds  $\tau$  for MOCC<sub>RNI</sub>.

### 6.4.2 Comparing MOCC<sub>RNI</sub> with MOCC<sub>PP</sub>

We now take a closer look at the performance of MOCC<sub>RNI</sub> versus that of MOCC<sub>PP</sub>. As combining MOCC with volatile nodes aims at improving the performance of MOCC for write-heavy scenarios, we focus on the WI and BA workloads.

First, we investigate how well MOCC<sub>PP</sub> and MOCC<sub>RNI</sub> handle skewed workloads with hotspots, i.e., nodes that are accessed very frequently. The first two columns of Figure 8 depict the results for varying the skewedness (determined by the parameter  $s$  of the Zipfian distribution). In the first column, we alter the writer skew  $s_w$ , in the second column the reader skew  $s_r$ . We can see clearly, that MOCC<sub>PP</sub> cannot cope with high writer skew at all. As soon as  $s_w$  increases beyond 0.5, the performance of MOCC<sub>PP</sub> deteriorates drastically. Due to the high contention, MOCC<sub>PP</sub> switches to a pessimistic lock-based protocol. This keeps transactions from aborting, but introduces waiting times for the release of locks, because a lot of transactions want to access the same data items in a skewed workload. The only case for which MOCC<sub>PP</sub> performs better is a uniformly distributed workload on the synthetic dataset. However, this case is the least relevant one in practice: real-world workloads are rarely uniformly distributed. The picture changes, when we look at the reader skew  $s_r$  (second column of Figure 8). MOCC<sub>RNI</sub>'s performance degrades slightly for a higher reader skew, as the skewed read operations traverse unproductive nodes more often. MOCC<sub>PP</sub>, on the other hand, shows constant performance for the synthetic dataset and even profits a bit for the Dell dataset. Nevertheless, MOCC<sub>RNI</sub> maintains an edge over MOCC<sub>PP</sub>.

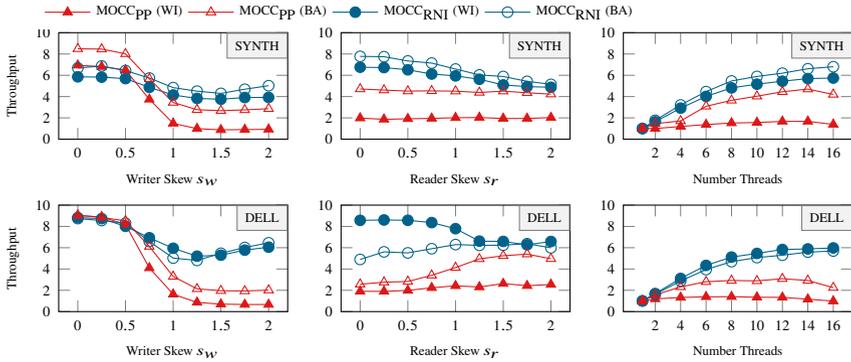


Fig. 8: MOCC<sub>RNI</sub> has a higher throughput than MOCC<sub>PP</sub>.

Second, we illustrate how MOCC<sub>RNI</sub> and MOCC<sub>PP</sub> compare for different degrees of concurrency (third column in Figure 8,  $s_W$  and  $s_R$  are set to the default value of 1). We increase the number of transactions that are running concurrently to see how well the two approaches can adapt to higher levels of concurrency. MOCC<sub>RNI</sub> scales much better, since it avoids many path conflicts from the outset with the use of volatile nodes.

## 6.5 Comparison with Deferred Node Deletions

In the next set of experiments, we compare RNI with an approach that defers node deletions as proposed by Lomet et al. [Lo04, LS97]. We implement deferred node deletions in PP as follows. When a user transaction attempts to delete an index node, the indexed property is removed (so that query results are correct) but the node deletion is deferred and the node is added to a queue. A background process periodically polls this queue and attempts to batch-prune the queued index nodes. If a background transaction fails due to a conflict, the index nodes are re-enqueued. We call this approach DeferredPP. Table 3 shows a comparison of the (normalized) throughput of DeferredPP with RNI. We conducted these experiments in Oak.

| workload<br>approach | SYNTH       |      |      | DELL        |      |             |
|----------------------|-------------|------|------|-------------|------|-------------|
|                      | WI          | BA   | RI   | WI          | BA   | RI          |
| DeferredPP           | <b>2.90</b> | 2.44 | 1.70 | <b>4.62</b> | 4.41 | 4.13        |
| RNI (optimized)      | <b>4.69</b> | 4.00 | 4.48 | <b>7.51</b> | 6.39 | 7.42        |
| RNI ( $\tau = 10$ )  | <b>4.15</b> | 4.00 | 3.74 | 6.73        | 6.31 | <b>7.07</b> |

Tab. 3: Comparison of normalized throughput between deferred node deletion and RNI.

The first interesting observation is that DeferredPP’s performance goes down with an increasing ratio of read transactions. For read-intensive workloads, deferring the deletions comes with a drawback. Essentially, the nodes scheduled for deletion are unproductive

nodes that have to be traversed by queries, driving down the query performance. The more read transactions we have, the more pronounced this effect is. More generally, when pruning a batch of nodes in background transactions, these transactions can clash with other transactions running in the system. While we always roll back a background transaction in a conflict (i.e., the regular transactions have precedence), this still consumes system resources and further reduces the throughput. Thus, DeferredPP is worse than RNI for all workloads. A scenario for which DeferredPP could potentially work is a system with write-intensive workloads that experiences phases of calm with a light load, e.g. during the night, in which the pruning takes place with a low probability of causing conflicts.

## 7 Conclusion

We investigated a problem that property-and-path (PP) indexes are faced with in hierarchical databases: the occurrence of path conflicts in the index when nodes with the same property (on different paths but with common ancestors in the database) are concurrently inserted and deleted. While the operations in the database go ahead without any issues, due to the propagation of deletes to ancestor nodes in the index, this causes a conflict and aborts the whole transaction. However, these aborts are preventable by leaving volatile nodes, i.e., nodes that are frequently inserted and deleted, in the index.

We propose the robust node index (RNI) that detects volatile nodes and prevents path conflicts due to the propagation of deletes. However, leaving volatile nodes in the index has a cost attached to it. The index becomes larger than it has to be and traversing additional, unproductive nodes during query evaluation has a negative impact on the performance. We experimentally evaluated the tradeoff between reducing the number of aborts and increasing query execution time and show how to tune RNI to maximize the throughput. This is done by only keeping volatile nodes in the index if their volatility is above a threshold  $\tau$ , i.e., if a node is inserted and deleted more than  $\tau$  times during a certain timeframe. Comparisons with other approaches, such as MOCC [WK16] and deferred delete [Lo04, LS97], confirm that RNI is able to significantly reduce the abort ratio from around 50% to below 10% for write-heavy workloads, thereby increasing the throughput up to a factor of five.

## Bibliography

- [Ad23] Adobe: , Adobe Experience Manager. <https://www.adobe.com/marketing-cloud/experience-manager.html>, 2023. [Online; accessed January 2023].
- [Ap17] Appuswamy, Raja; Anadiotis, Angelos; Porobic, Danica; Iman, Mustafa; Ailamaki, Anastasia: Analyzing the Impact of System Architecture on the Scalability of OLTP Engines for High-Contention Workloads. *PVLDB*, 11(2):121–134, 2017.
- [Ap22] Apache: , Apache Jackrabbit Oak. <https://jackrabbit.apache.org/oak/>, 2022. [Online; accessed January 2023, last updated November 2022].

- [Be95] Berenson, Hal; Bernstein, Philip A.; Gray, Jim; Melton, Jim; O'Neil, Elizabeth J.; O'Neil, Patrick E.: A Critique of ANSI SQL Isolation Levels. In: SIGMOD. 1995.
- [Be11] Bernstein, Philip A.; Reid, Colin W.; Wu, Ming; Yuan, Xinhao: Optimistic Concurrency Control by Melding Trees. PVLDB, 4(11), 2011.
- [Be15] Bernstein, Philip A.; Das, Sudipto; Ding, Bailu; Pilman, Markus: Optimizing Optimistic Concurrency Control for Tree-Structured, Log-Structured Databases. SIGMOD, 2015.
- [CH10] Cormode, Graham; Hadjieleftheriou, Marios: Methods for finding frequent items in data streams. The VLDB Journal, 19(1):3–20, 2010.
- [CMS02] Chung, Chin-Wan; Min, Jun-Ki; Shim, Kyuseok: APEX: An adaptive path index for XML data. In: SIGMOD. ACM, pp. 121–132, 2002.
- [Co01] Cooper, Brian F.; Sample, Neal; Franklin, Michael J.; Hjaltason, Gísli R.; Shadmon, Moshe: A Fast Index for Semistructured Data. In: VLDB. 2001.
- [DR13] Diegues, Nuno Lourenco; Romano, Paolo: Bumper: Sheltering Transactions from Conflicts. In: IEEE SRDS. 2013.
- [Fi02] Fiebig, Thorsten; Helmer, Sven; Kanne, Carl-Christian; Moerkotte, Guido; Neumann, Julia; Schiele, Robert; Westmann, Till: Anatomy of a native XML base management system. VLDB J., 11(4):292–314, 2002.
- [Fi13] Finis, Jan; Brunel, Robert; Kemper, Alfons; Neumann, Thomas; Färber, Franz; May, Norman: DeltaNI: An Efficient Labeling Scheme for Versioned Hierarchical Data. In: SIGMOD. pp. 905–916, 2013.
- [Gr14] Graefe, Goetz; Halim, Felix; Idreos, Stratos; Kuno, Harumi A.; Manegold, Stefan; Seeger, Bernhard: Transactional support for adaptive indexing. VLDB J., 23(2), 2014.
- [GW97] Goldman, Roy; Widom, Jennifer: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: VLDB. 1997.
- [Ha17] Harding, Rachael; Van Aken, Dana; Pavlo, Andrew; Stonebraker, Michael: An Evaluation of Distributed Concurrency Control. PVLDB, 2017.
- [HHL06] Haustein, Michael Peter; Härder, Theo; Luttenberger, Konstantin: Contest of XML Lock Protocols. In: VLDB. 2006.
- [HL11] Haw, Su-Cheng; Lee, Chien-Sing: Data storage practices and query processing in XML databases: A survey. Knowledge-Based Systems, 24(8):1317–1340, 2011.
- [Id11] Idreos, Stratos; Manegold, Stefan; Kuno, Harumi A.; Graefe, Goetz: Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores. PVLDB, 4(9):585–597, 2011.
- [JPA09] Johnson, Ryan; Pandis, Ippokratis; Ailamaki, Anastasia: Improving OLTP Scalability using Speculative Lock Inheritance. PVLDB, 2(1):479–489, 2009.
- [Ki15] Kimura, Hideaki: FOEDUS: OLTP Engine for a Thousand Cores and NVRAM. In: SIGMOD. 2015.
- [Lo04] Lomet, David B.: Simple, Robust and Highly Concurrent B-trees with Node Deletion. In: ICDE. pp. 18–27, 2004.

- [Lo15] Loro, Alessandra; Gruenheid, Anja; Kossmann, Donald; Profeta, Damien; Beaudequin, Philippe: Indexing and Selecting Hierarchical Business Logic. *PVLDB*, 8(12):1656–1667, 2015.
- [LS97] Lomet, David B.; Salzberg, Betty: Concurrency and Recovery for Index Trees. *VLDB J.*, 6(3):224–240, 1997.
- [LY81] Lehman, Philip L.; Yao, s. Bing: Efficient Locking for Concurrent Operations on B-trees. *ACM TODS.*, 6(4):650–670, December 1981.
- [Ma15] Mathis, Christian; Härder, Theo; Schmidt, Karsten; Bächle, Sebastian: XML indexing and storage: fulfilling the wish list. *Computer Science - R&D*, 30(1), 2015.
- [Ni06] Nicolaisen, Thomas Ferris: The Use of Open Source and Open Standards in Web Content Management Systems. Master’s thesis, University of Oslo, Oslo, Norway, May 2006.
- [RFA16] Ren, Kun; Faleiro, Jose M.; Abadi, Daniel J.: Design Principles for Scaling Multi-core OLTP Under High Contention. In: *SIGMOD*. 2016.
- [RTA14] Ren, Kun; Thomson, Alexander; Abadi, Daniel J.: An Evaluation of the Advantages and Disadvantages of Deterministic Database Systems. *PVLDB*, 2014.
- [Sh15] Shukla, Dharna; Thota, Shireesh; Raman, Karthik; Gajendran, Madhan; Shah, Ankur; Ziuzin, Sergii; Sundaram, Krishnan; Guajardo, Miguel Gonzalez; Wawrzyniak, Anna; Boshra, Samer; Ferreira, Renato; Nassar, Mohamed; Koltachev, Michael; Huang, Ji; Sengupta, Sudipta; Levandoski, Justin J.; Lomet, David B.: Schema-Agnostic Indexing with Azure DocumentDB. *PVLDB*, 2015.
- [TA10] Thomson, Alexander; Abadi, Daniel J.: The Case for Determinism in Database Systems. *PVLDB*, 2010.
- [Th12] Thomson, Alexander; Diamond, Thaddeus; Weng, Shu-Chun; Ren, Kun; Shao, Philip; Abadi, Daniel J.: Calvin: Fast Distributed Transactions for Partitioned Database Systems. In: *SIGMOD*. 2012.
- [Ti18] Tian, Boyu; Huang, Jiamin; Mozafari, Barzan; Schoenebeck, Grant: Contention-Aware Lock Scheduling for Transactional Databases. *PVLDB*, 2018.
- [TYJ09] Tzoumas, Kostas; Yiu, Man Lung; Jensen, Christian S.: Workload-Aware Indexing of Continuously Moving Objects. *PVLDB*, 2009.
- [WBH20] Wellenzohn, Kevin; Böhlen, Michael H.; Helmer, Sven: Dynamic Interleaving of Content and Structure for Robust Indexing of Semi-Structured Hierarchical Data. *Proc. VLDB Endow.*, 13(10):1641–1653, 2020.
- [WK16] Wang, Tianzheng; Kimura, Hideaki: Mostly-optimistic Concurrency Control for Highly Contended Dynamic Workloads on a Thousand Cores. *PVLDB*, 10(2):49–60, 2016.
- [YC16] Yan, Cong; Cheung, Alvin: Leveraging Lock Contention to Improve OLTP Application Performance. *PVLDB*, 9(5):444–455, 2016.
- [Yu16] Yuan, Yuan; Wang, Kaibo; Lee, Rubao; Ding, Xiaoning; Xing, Jing; Blanas, Spyros; Zhang, Xiaodong: BCC: Reducing False Aborts in Optimistic Concurrency Control with Low Cost for In-memory Databases. *PVLDB*, 9(6):504–515, 2016.

# Tuning Cassandra through Machine Learning

Florian Eppinger<sup>1</sup> Uta Störl<sup>2</sup>

**Abstract:** The distributed nature and scalability of NoSQL databases make them an ideal data storage repository for a variety of use cases. While NoSQL databases are delivered with a default “off-the-shelf” configuration, they offer configuration settings to adjust a database’s behavior to a specific use case and environment. The abundance and oftentimes imperceptible inter-dependencies of configuration settings make it difficult to optimize and performance-tune a NoSQL system. This work explores Machine Learning as a means to automatically tune a NoSQL database for optimal performance. Using Ensemble Machine Learning algorithms, multiple ML models were fitted with a training dataset that incorporates properties of the NoSQL physical configuration (replication and sharding). The best models were then employed as surrogate models to optimize the Database Management System’s configuration settings for best performance using a Black-box Optimization algorithm. Multiple experiments were carried out with an Apache Cassandra database to demonstrate the feasibility of this approach, even across varying physical configurations. The tuned Database Management System configurations yielded throughput improvements of up to 4%, read latency reductions of up to 43%, and write latency reductions of up to 39% when compared to the default configuration settings.

**Keywords:** AI for Database Systems; NoSQL; Machine Learning; Performance Modeling; Tuning; Black-box Optimization

## 1 Introduction

The rate at which data is created, used, and persisted increases rapidly. While Relational Database Management Systems (RDBMSs) continue to play an important role in today’s technology environments, Non-relational SQL or Non-SQL (NoSQL) Databases (DBs) have become an integral part of real-time analytics or big data applications [CL19; SF12]. Choosing the best NoSQL solution and developing the best physical design for a given use case can be a challenging task on its own [HAR16; Lo15; QCH18]. Furthermore, NoSQL technologies offer an abundance of configuration settings that allow the system administrator to adjust the DB behavior to further meet the requirements of a particular use case. Many of the configuration parameters have an impact on the performance of the NoSQL DB, i.e., its throughput and latency.

*Finding the configuration that maximizes throughput or minimizes latency for a given use case is complex, and DB behavior is not always self-explanatory* as shown by Preuveneers; Joosen [PJ20].

---

<sup>1</sup> University of Hagen, Databases and Information Systems, Hagen, Germany [florian.eppinger@gmail.com](mailto:florian.eppinger@gmail.com)

<sup>2</sup> University of Hagen, Databases and Information Systems, Hagen, Germany [uta.stoerl@fernuni-hagen.de](mailto:uta.stoerl@fernuni-hagen.de)

Having the ability to predict performance measures for varying workloads and physical configurations, and to optimize Database Management System (DBMS) configuration settings accordingly, could be beneficial due to a variety of reasons, including sudden changes in user behavior, application changes, and changes to the hardware environment.

The methodology and results outlined in this study consider aspects of the physical configuration *and* the DBMS configuration settings during tuning, and we thus make the following contributions:

- We analyze the quality of Machine Learning (ML) models that are trained to predict performance metrics for varying workloads, physical configurations, and DBMS configurations.
- We measure the influence of the training dataset size on the quality of these ML models.
- We evaluate how features representing the physical configuration impact the quality of the ML models.
- We explore the ability to tune the DBMS configuration settings for a specific physical configuration using the ML models and Black-box Optimization (BBO).

Section 2 reviews related work. Section 3 briefly introduces the end-to-end methodology and the training dataset. Results and findings are presented and evaluated in section 4. The document concludes with section 5, which discusses the results and suggests areas of future work.

## 2 Related Work

A variety of proposals have been made to mitigate the performance tuning challenges outlined in section 1 through automation. Some of these methods are able to tune a variety of software solutions in a generic fashion [Wa18; Zh17], while others are geared specifically toward RDBMS or NoSQL technology. This work can be categorized into solutions that tune DB performance via the DBMS configuration settings [Ak21; KAS15; Xi17; Zh19] and solutions that tune DB performance via the physical design of the DB [Be15; Cr13; Fa16]. Tuning Methods include Control Theory, Expert Systems, and a variety of Machine Learning algorithms.

The main difference between this paper and related performance-tuning work for NoSQL systems is that the Tuning Domain includes both DBMS configuration settings *and* the DB physical design in form of sharding and replication. While Preuveneers and Joosen consider both aspects [PJ20], their technique differs in several ways. First, Preuveneers and Joosen utilize multiple predefined tactics in an attempt to tune the NoSQL system *online*. Second, the authors utilize the ML model to map DB scenarios consisting of workload metrics, resource utilization, physical configuration, etc. to an ideal DBMS configuration and DB physical design. On the other hand, this study evaluates ML techniques to fit ensemble models to

predict DB performance. These models are then used by BBO algorithms to find an optimum DBMS configuration for a given workload and physical configuration. Consequently, the methods used in this paper are related much closer to the approach presented by Xiong et al. [Xi17], except that this work includes features for the physical configuration, utilizes Apache Cassandra instead of HBase, and employs a different optimization algorithm. For an in-depth study of related work and a point-by-point comparison refer to the extended version of this paper [ES22].

### 3 Methodology

Apache Cassandra<sup>3</sup> (“Cassandra”) was selected as the basis for the experiments. A tuning domain was defined, and a training dataset was generated using a sample database. This training dataset was then transformed into input for Decision Tree (DT) ML algorithms to fit various ML models. The models’ objective is to accurately predict performance metrics of the database for a given workload, DBMS configuration, and physical design. The performance of the ML models was evaluated using commonly accepted quality measures. The predictions of the ML models were then used to find optimized DBMS configuration values for a given workload and physical configuration using a Black-box Optimization algorithm. Finally, actual performance of the optimized configurations was evaluated and compared against the DB performance of the default DBMS configuration.

#### 3.1 Tuning Domain

This section describes the Tuning Domain (TD), i.e., the workload properties, the specific DBMS configuration settings, as well as aspects of the Cassandra physical configuration that were considered within the scope of this study:

- *Workload*: To follow the approach chosen in related work, such as [Cr13] and [PJ20], three workloads were included:
  - 50% read using the primary key, 50% write (*readwrite*)
  - 95% read using the primary key, 5% write (*readheavy*)
  - 5% read using the primary key, 95% write (*writeheavy*)
- *DBMS Configuration*: To reduce the complexity of this study, a subset of performance-relevant Apache Cassandra configuration settings was selected through research and targeted experiments.
- *Physical Design*: to account for aspects of the physical configuration, we considered both *sharding* and *replication*.

Table 1 provides a summary of the features and feature domains that were considered in this study.

---

<sup>3</sup> <https://cassandra.apache.org/> (visited on Sept. 19<sup>th</sup>, 2022)

| Feature                      | Feature Category   | Domain  |
|------------------------------|--------------------|---|
| wl_read_%                    | Workload           | (0% – 100%)   |
| wl_write_%                   | Workload           | (0% – 100%)   |
| trickle_fsync                | DBMS Configuration | {true, false}   |
| key_cache_size_in_mb         | DBMS Configuration | {0, 2 <sup>0</sup> , ..., 2 <sup>5</sup> }            |
| row_cache_size_in_mb         | DBMS Configuration | {0, 20, 40, 60, ..., 200}                             |
| commitlog_segment_size_in_mb | DBMS Configuration | {2 <sup>2</sup> , ..., 2 <sup>6</sup> }               |
| concurrent_reads             | DBMS Configuration | {2 <sup>1</sup> * disks, ..., 2 <sup>5</sup> * disks} |
| concurrent_writes            | DBMS Configuration | {2 <sup>1</sup> , ..., 2 <sup>8</sup> }               |
| memtable_heap_space_in_mb    | DBMS Configuration | {2 <sup>-5</sup> * heap, ..., 2 <sup>-1</sup> * heap} |
| node_count (n)               | Physical Design    | {2, 3, 4}   |
| replication_factor (rf)      | Physical Design    | {1, 2, 3, 4}  |

Tab. 1: Features included in the Tuning Domain

### 3.2 Tuning Subdomains

In this study, we analyzed the impact of the workload, DBMS configuration, and the physical design on the accuracy of the ML models and their ability to serve as a surrogate model for performance tuning. We, therefore, defined TD1-TD4 as subsets of the TD. As shown in table 2, these *Tuning Subdomains* focus on specific aspects of the feature set.

| TD         | Physical Config.  | Workload                          | DBMS Config. |
|------------|-------------------|-----------------------------------|--------------|
| <i>TD1</i> | *                 | *                                 | *            |
| <i>TD2</i> | *                 | wl_read_%=fixed, wl_write_%=fixed | *            |
| <i>TD3</i> | n=fixed, rf=fixed | *                                 | *            |
| <i>TD4</i> | n=fixed, rf=fixed | wl_read_%=fixed, wl_write_%=fixed | *            |

Tab. 2: Definition of the tuning subdomains

TD1 represents the entire tuning domain, i.e. all of the features introduced in Table 1. TD2 reduces complexity by removing the workload features from the feature vector. It was designed to train ML models that are able to make predictions for a fixed workload. TD3, on the other hand, excludes the physical configuration features from the feature vector. TD4 only considers the DBMS configuration settings.

### 3.3 Tuning Methodology

The methodology applied in this study used ensemble-based ML algorithms to develop models that are able to make DB throughput and latency predictions. Leveraging the ML models' predictions, a BBO algorithm is then utilized to search the optimum DBMS configuration for a given workload and DB physical configuration.

*Ensemble methods* combine multiple ML models to make a final prediction. Two popular ensemble methods are *Bootstrap Aggregation (Bagging)* [Br01] and *Boosting* [Fr01; NK13], which were explored in form of Random Forest (RF) and Gradient Boosting Decision Tree (GBDT) ML algorithms for various reasons that are outlined in the extended version of this paper [ES22].

The fitted ML model can be considered a surrogate model that can respond with a performance prediction for a given set of inputs. Because the model itself is a black-box and does not expose any meaningful information that could be used to find an optimum using a gradient-based optimization approach, this study utilized BBO as a means to find an optimum configuration using the ML model's performance predictions. Various BBO algorithms were explored. However, the results presented in this paper are based on the Simulated Annealing algorithm, which was shown to be the most efficient among the ones evaluated [ES22]. The algorithm is similar to Local Optimization except that it attempts to avoid getting stuck in a non-global optimum by adding a random element for further exploration of the domain [Jo89].

### 3.4 Training Data

A new dataset consisting of 32,757 training examples was created. A training example comprises workload properties, the values for DBMS configuration settings, a representation of the DB physical configuration, as well as the actual performance values that were measured when the workload was executed against the DB using this particular DBMS and physical configuration. Additional details about the training dataset can be found in the extended version of this study [ES22].

## 4 Experimental Evaluation

The following results and findings were gathered during experiments that were carried out on a DB cluster with 5 nodes provided by the University of Hagen. Please refer to the extended version of this paper [ES22] for additional details regarding the implementation as well as a more in-depth analysis of results beyond the DB read and write latencies that this paper focuses on.

### 4.1 ML Model Quality

Table 3 lists the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) for DB throughput and latencies. The measurements were established by training a total of 6 individual ML models using TD1, i.e. the entire training dataset as defined in section 3.2.

|                | Overall Throughput |           | Read Latency |       | Write Latency |       |
|----------------|--------------------|-----------|--------------|-------|---------------|-------|
|                | RF                 | GBDT      | RF           | GBDT  | RF            | GBDT  |
| <b>MAE</b>     | 2,810.940          | 2,880.110 | 0.307        | 0.279 | 0.203         | 0.189 |
| <b>MAE (%)</b> | 3.290              | 3.370     | 2.940        | 2.730 | 3.030         | 2.890 |
| <b>RMSE</b>    | 4,646.420          | 5,064.020 | 0.500        | 0.493 | 0.369         | 0.369 |

Tab. 3: ML Model Quality

It should be noted that the hyperparameters of the ML models referenced in this study were tuned for each ML model individually. Using a random split methodology, 75% of the dataset was used to fit the models with holdout validation (see [Gé17]), and the remaining 25% were used for testing.

To understand the correlation between dataset size and ML model prediction accuracy, various experiments were conducted with artificially reduced datasets. A total of 9 ML models were trained for each of the performance measures using the following dataset sizes: 128, 256, 512, 1,024, 2,048, 4,096, 8,192, 16,384, 24,672. To ensure a fair evaluation and reduce the chance of randomly selecting a non-representative test distribution, the test dataset was kept at a fixed size of 8,000 examples. The results are shown in figure 1. A training dataset size of 128 examples yielded a GBDT model that predicted overall throughput with an MAE of 7,326 and a RF model that predicted overall throughput with an MAE of 9,973. Increasing the training dataset size to just 1,024 records significantly reduced the MAE values to 3,903 (GBDT) and 4,305 (RF). Additional accuracy could be gained by further increasing the dataset size. However, only minor improvements could be seen for read and write latencies with datasets exceeding 4,096 examples.

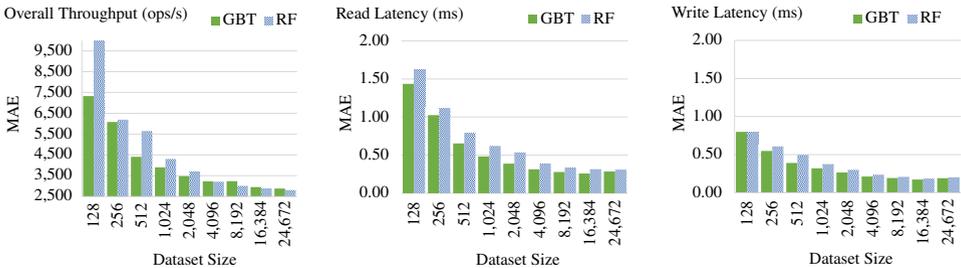


Fig. 1: Influence of the dataset size on the prediction accuracy of the ML models

To compare the TD1 ML model quality to the less complex Tuning Domains, ML models were trained for TD2-TD4 to predict write latencies. Four individual models were fitted for each combination of TD and ML algorithm using datasets with 128, 256, 512, and 1,024 training examples. For this experiment, a test dataset size of 250 was used to determine the MAE values for each of the models. The results are shown in figure 2 and confirm that the ML models trained with TD1 performed worse compared to TD2-TD4 when identically sized training datasets were used. A GBDT model trained with 128 randomly

selected examples from the TD1 dataset yielded an MAE of 1.42, which represents an error percentage of 13.62% compared to 0.65 (5.88%) for TD2. The MAE of the GBDT model dropped to 0.39 (3.55%) with 512 training examples for TD2, while it took four times the number of training examples to reach comparable accuracy within TD1. It is also worth noting that with 1,024 training examples, the TD1 and TD3 models were of similar quality despite the added complexity of the TD1 model that is able to make predictions for eight different physical configurations.



Fig. 2: Influence of the dataset size on the write latency prediction accuracy of the ML models trained with different Tuning Domains

## 4.2 Tuning performance with Black-box Optimization

The Simulated Annealing BBO algorithm was then used to optimize the DBMS configuration settings in a variety of experiments that involved the fitted ML models. The configurations proposed by the algorithm were then applied to the Cassandra cluster, workloads were executed, and results were captured.

First, we optimized DBMS configuration settings for various workloads with a fixed physical configuration ( $n=4$ ,  $rf=3$ ) using the fitted TD1 model. In addition to the *readwrite* (50% read, 50% write), *readheavy* (95% read, 5% write) and *writeheavy* (5% read, 95% write) workloads, experiments were conducted with a workload that differed from the workloads used to train the ML model (25% read, 75% write). For each performance measure, five independent experiments were carried out for each proposed DBMS configuration and workload, and the average performance was calculated. Results for the actual read and write latencies are shown in figure 3. Tuned configurations reduced the read latency by more than 42% for write-heavy workloads and the write latency by more than 39% for workloads with a significant amount of read operations. It should be noted that configurations that improved read latency resulted in a higher write latency and vice versa, reducing overall throughput.

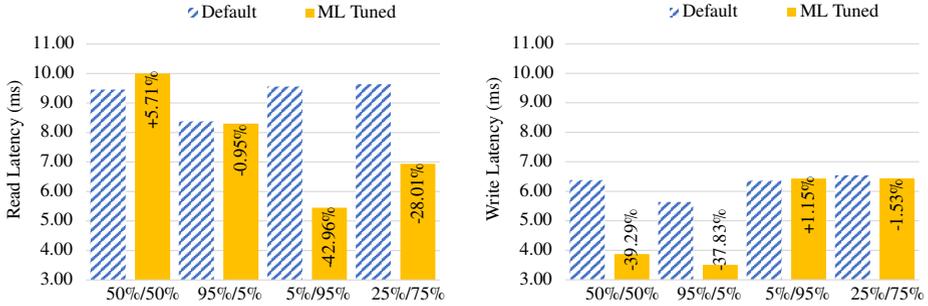


Fig. 3: Actual latency measurements for default and ML/BBO-tuned DBMS configurations

However, the results clearly demonstrate that the ML model was able to successfully derive information regarding the DBMS configuration settings’ impact on the DB performance.

Next, we applied the optimization method to different physical configurations. This time the objective of the algorithm was to tune the performance by adjusting the DBMS configuration for varying physical configurations. The measurements also included a physical configuration with two nodes and a replication factor of one to analyze how well the methodology works for previously unseen physical configurations. Figure 4 compares the write latency of the ML-tuned configuration to the default configuration (*readwrite* workload). The results demonstrate that the ML-based tuning method successfully tuned the DB latency for a variety of physical configurations. However, it also highlights that it failed to optimize

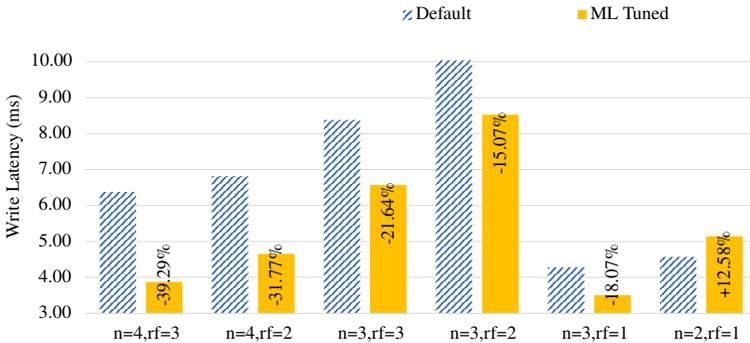


Fig. 4: Write latencies for default and ML/BBO-tuned DBMS configurations under varying physical configurations

performance for the physical configuration that was not previously encountered during the training phase. The write latency increased by 12.58% from 4.57 *Milliseconds*(ms) to 5.14ms. It is possible that the ML algorithms failed to extract and derive a meaningful trend or that the ordinal encoding method for the corresponding features did not cultivate this kind of predictive quality in the model.

We also analyzed models that were trained with the tuning subdomains TD2-TD4. Section 4.1

showed significant accuracy differences depending on what tuning domain was used to fit the ML models. To evaluate the ability to tune the DBMS configurations with these models, the Simulated Annealing algorithm was used to search for an optimum DB configuration for each of the TDs using the ML models fitted with 1,024 training examples. The actual DB performance was then measured for the proposed configurations. This process was repeated three times for each TD, and average performance results were calculated. The goal of the experiment was to optimize read and write latency for the *readheavy* workload (95% read/5% write). The results are shown in figure 5.

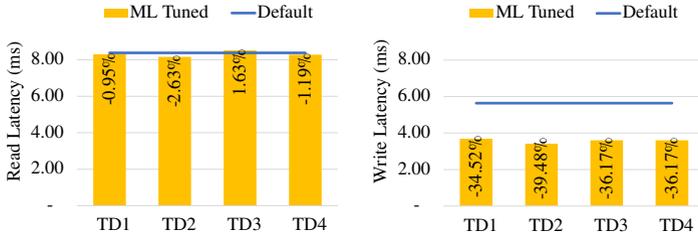


Fig. 5: Actual latency measurements for DBMS configurations that were optimized with ML models trained with TD1-TD4

The superior accuracy of TD2 and TD4 models did not result in better DBMS configuration proposals. We determined that the model’s tuning ability depended both on its accuracy as well as the quality of the training data. Because the training subset was chosen randomly, it contained examples with poor performance. The ML models for the less complex TDs were able to make more accurate predictions, but this did not help the BBO algorithm find a better configuration if they were not fitted with high-performance training examples.

## 5 Discussion

This study evaluated Machine Learning and Black-box Optimization as a means to performance-tune NoSQL DBs for varying physical configurations and workloads. When fitted with the entire dataset, the ML models yielded an MAE of 2.73% and 2.89% for read and write latencies, respectively. Omitting the workload from the feature set (TD2) significantly improved accuracy, while omitting the features representing the physical configuration (TD3) resulted in moderate improvements only. This observation may lead to the conclusion that *the physical configuration adds less complexity to the model than varying workloads*. When using training datasets of 1,024 examples, the MAE of the TD1 and TD3 GBDT models were almost identical when predicting read latencies, thereby implying that it is more efficient to train a single predictive model for multiple physical configurations than it is to train an individual model for each physical configuration.

The most accurate models were then used to optimize DBMS configuration settings for a given workload and physical configuration using BBO. The algorithm was able to find configurations for improved performance in most situations. For a physical design with four NoSQL nodes and a replication factor of three, latencies could be reduced by up to

42.96% (read) and 39.29% (write) depending on the workload composition. Similar results were achieved for varying physical configurations. Additional experiments showed that DB latency could be improved even for previously unseen workloads.

While the tuned DBMS configurations did result in performance improvements, none of the results matched or exceeded the performance of the best-performing training examples. As an example, the BBO-tuned DBMS configuration ( $n=4$ ,  $rf=3$ , writeheavy) resulted in maximum throughput of 103,965 operations per second (op/s) compared to 95,240 op/s for the default configuration. However, the most performant training example resulted in 111,018 op/s, implying an additional tuning potential of 6.5%. A potential explanation for this is the generalization capability of the ML model, which enables the ML model to make more accurate predictions for previously unseen DBMS configurations but also regulates outlier configurations. These outliers represent subpar and also optimum configurations.

Several discoveries and choices were made regarding technology and methodology. Furthermore, several areas remained unexplored due to time and resource constraints. The following list reflects on some of these items and highlights potential areas for future work:

- This study targeted individual DB performance measures. Performance objectives vary, and it may be necessary to meet multiple performance goals. Xiong et al. approach this by combining multiple weighted performance measures into a single optimization objective [Xi17], an attempt that could be explored further.
- It was also noted that the training dataset captures performance results that are specific to the hardware environment and technologies used. Exploring the transformation or scaling of training examples or derived knowledge for a different environment or technology stack appears worthwhile.
- The tuning domain of this study is limited. More aspects exist, including indexes, schema design, consistency levels, etc., that could be evaluated in more detail.
- Attempting to tune the DBMS configuration for previously unseen physical designs did result in performance that under-performed the default DBMS configuration. The corresponding features were encoded as ordinal values, and changing the encoding scheme may improve these results.
- The study treated DBMS configuration settings as global settings, i.e., the same configuration settings were used for all nodes of the Cassandra node ring. However, many settings can be configured individually for each cluster node. Research in this area presented by Cruz et al. [Cr13] could be evaluated as an extension to the methodology outlined in this document.
- This study evaluated RF and GBDT to develop a surrogate model for DB performance predictions. Other ML algorithms exist and may exhibit a better prediction quality. Similarly, various BBO algorithms exist, some of which have been shown to produce better results than the Simulated Annealing algorithms [Ch21]. A more systematic evaluation of different algorithms could be carried out.

## References

- [Ak21] Aken, D. V.; Yang, D.; Brillard, S.; Fiorino, A.; Zhang, B.; Bilien, C.; Pavlo, A.: An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems. *Proc. VLDB Endowment* 14/, pp. 1241–1253, 2021.
- [Be15] Bermbach, D.; Müller, S.; Eberhardt, J.; Tai, S.: Informed Schema Design for Column Store-Based Database Services. In: *Proc. SOCA 2015*. IEEE, pp. 163–172, Oct. 2015.
- [Br01] Breiman, L.: Random Forests. *Machine Learning* 45/, pp. 5–32, Jan. 2001.
- [Ch21] Chen, P.; Huo, Z.; Li, X.; Dou, H.; Zhu, C.: ConfAdvisor: An Automatic Configuration Tuning Framework for NoSQL Database Benchmarking with a Black-box Approach. In: *Bench 2020, Revised Selected Papers*. Vol. 12614, Springer, pp. 106–124, 2021.
- [CL19] Chen, J. K.; Lee, W. Z.: An introduction of NoSQL databases based on their categories and application industries. *Algorithms* 12/, May 2019.
- [Cr13] Cruz, F.; Maia, F.; Matos, M.; Oliveira, R.; Paulo, J.; Pereira, J.; Vilaça, R.: MeT: Workload aware elasticity for NoSQL. In: *Proc. EuroSys 2013*. Pp. 183–196, 2013.
- [ES22] Eppinger, F.; Störl, U.: NoSQL Database Tuning through Machine Learning. *CoRR abs/2212.12301*, 2022, arXiv: 2212.12301, URL: <http://arxiv.org/abs/2212.12301>.
- [Fa16] Farias, V. A.; Sousa, F. R.; Maia, J. G.; Gomes, J. P.; MacHado, J. C.: Machine Learning Approach for Cloud NoSQL Databases Performance Modeling. In: *Proc. CCGrid 2016*. IEEE, pp. 617–620, July 2016.
- [Fr01] Friedman, J. H.: Greedy Function Approximation: A Gradient Boosting Machine. *Source: The Annals of Statistics* 29/, pp. 1189–1232, 2001.
- [Gé17] Géron, A.: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2017.
- [HAR16] Herrero, V.; Abelló, A.; Romero, O.: NOSQL design for analytical workloads: Variability matters. In: *Proc. ER 2016*. Vol. 9974, Springer, pp. 50–64, 2016.
- [Jo89] Johnson, D. S.; Aragon, C. R.; Mcgeoch, L. A.; Schevon, C.: Optimization By Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Oper. Res.* 37/, pp. 865–892, 1989.
- [KAS15] Khattab, A.; Algergawy, A.; Sarhan, A.: MAG: A performance evaluation framework for database systems. *Knowledge-Based Systems* 85/, pp. 245–255, Sept. 2015.

- [Lo15] Lourenço, J. R.; Cabral, B.; Carreiro, P.; Vieira, M.; Bernardino, J.: Choosing the right NoSQL database for the job: a quality attribute evaluation. *Journal of Big Data* 2/, Dec. 2015.
- [NK13] Natekin, A.; Knoll, A.: Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics* 7/, 2013.
- [PJ20] Preuveneers, D.; Joosen, W.: Automated configuration of NoSQL performance and scalability tactics for data-intensive applications. *Informatics* 7/, Aug. 2020.
- [QCH18] Qader, M. A.; Cheng, S.; Hristidis, V.: A comparative study of secondary indexing techniques in LSM-based NoSQL databases. In: *Proc. SIGMOD 2018*. ACM, pp. 551–566, May 2018.
- [SF12] Sadalage, P. J.; Fowler, M.: *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.
- [Wa18] Wang, S.; Li, C.; Hoffmann, H.; Lu, S.; Sentosa, W.; Kistijantoro, A. I.: Understanding and auto-adjusting performance-sensitive configurations. In: *Proc. ASPLOS 2018*. Vol. 53, ACM, pp. 154–168, Mar. 2018.
- [Xi17] Xiong, W.; Bei, Z.; Xu, C.; Yu, Z.: ATH: Auto-Tuning HBase’s Configuration via Ensemble Learning. *IEEE Access* 5/, pp. 13157–13170, June 2017.
- [Zh17] Zhu, Y.; Liu, J.; Guo, M.; Bao, Y.; Ma, W.; Liu, Z.; Song, K.; Yang, Y.: BestConfig: Tapping the performance potential of systems via automatic configuration tuning. In: *Proc. SoCC 2017*. ACM, pp. 338–350, Sept. 2017.
- [Zh19] Zhang, J.; Liu, Y.; Zhou, K.; Li, G.; Xiao, Z.; Cheng, B.; Xing, J.; Wang, Y.; Cheng, T.; Liu, L.; Ran, M.; Li, Z.: An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning. In: *Proc. SIGMOD 2019*. ACM, pp. 415–432, June 2019.

# GTPC: Towards a Hybrid OLTP-OLAP Graph Benchmark

Muhammad Attahir Jibril<sup>1</sup>, Alexander Baumstark<sup>2</sup>, Kai-Uwe Sattler<sup>3</sup>

**Abstract:** Graph databases are gaining increasing relevance not only for pure analytics but also for full transactional support. Business requirements are evolving to demand analytical insights on fresh transactional data, thereby triggering the emergence of graph systems for hybrid transactional-analytical graph processing (HTAP). In this paper, we present our ongoing work on GTPC, a hybrid graph benchmark targeting such systems, based on the TPC-C and TPC-H benchmarks.

**Keywords:** Benchmarking; Graph HTAP; Graph Databases

## 1 Introduction

With the ever-growing amount of data from various real-world applications that lend themselves to being modelled as graphs, graph databases are receiving more and more attention from both the industry and academia for efficiently managing and processing such graph data. Various enterprises driven by their respective markets such as social media, logistics, e-commerce etc., are capitalizing on graph analytics for decision-support purposes. On top of that, in reality, the graph data is hardly without update operations. In fact, enterprises aim at continuously acquiring fresh business insights in order to make crucial business decisions. Some graph databases e.g. Neo4j [Neo4j], TigerGraph [TigerGraph], Ultipa [Ultipa] etc. support analytical workloads in addition to transactional workloads [Be19]. Additionally, more work is being put into graph storage systems such as LiveGraph [Zh20] that support concurrent execution of transactional and analytical workloads – towards the development of hybrid transactional-analytical (HTAP) graph databases.

Despite the relevance of emerging HTAP graph database systems, we identify the lack of a hybrid benchmark with mixed workloads that aims at these systems. Although there exist hybrid OLTP-OLAP benchmarks for other data models e.g. the relational model, however, they cannot be used directly for graphs. One such benchmark is the CH-benCHmark [Co11], which is based on the TPC-C [TPC-C10] and TPC-H [TPC-H21] benchmarks. In this paper, we propose GTPC, a hybrid graph benchmark modelled on a *product* graph (as obtained in domains like e-commerce) by adapting the underlying concepts of the CH-benCHmark. Firstly, GTPC employs schema optimizations to convert the relational CH-benCHmark schema into a property graph schema. Secondly, it provides a data generator (presently

---

<sup>1</sup> TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, muhammad-attahir.jibril@tu-ilmenau.de

<sup>2</sup> TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, alexander.baumstark@tu-ilmenau.de

<sup>3</sup> TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, kus@tu-ilmenau.de

a modified version of the CH-benCHmark data generator) for generating product graphs. Thirdly, it transforms the CH-benCHmark queries into transactional and analytical graph queries. Lastly, it specifies a mixed workload of the formulated graph queries. Although relational databases would traditionally be the solution for product graphs (i.e. products, orders and transactions data modelled as graphs), however, interestingly, product graphs are the most popular among graphs that model non-human entities [Sa20]. There is a need for benchmarks targeting such graph data and workloads, as none of the existing property graph benchmarks addresses the issue [Sa20]. This would provide a metric(s) with which to compare the systems used for such product graphs. Moreover, it would allow obtaining empirical insights into why industry practitioners use graph systems for processing data that would otherwise be considered suitable for relational systems.

To summarize, the increasing relevance of executing mixed workloads using HTAP graph databases as well as the huge adoption of graphs in modelling product data by industry practitioners motivate the need for a hybrid OLTP-OLAP graph benchmark. Such a benchmark is important as it serves as a useful tool for testing and comparing systems in terms of HTAP metrics such as freshness and performance isolation of concurrently running transactional and analytical workloads. Therefore, we present our ongoing work on benchmarking graph HTAP systems. Our contributions in this paper are as follows:

- We propose GTPC, a hybrid graph benchmark with mixed OLTP-OLAP workloads on a synthetically generated product graph based on the TPC-C and TPC-H benchmarks.
- Using our Poseidon graph database [Ji21] as an example, we implement and run the GTPC benchmark<sup>4</sup> to show its effectiveness in testing graph systems' handling of mixed workloads and revealing the performance interplay between analytical and transactional graph query workloads executed concurrently in a multi-user setting.

## 2 Related work

There exist various benchmarking solutions for graph databases stressing different workloads such as subgraph pattern matching, recursive path queries etc. These solutions include HPC Scalable Graph Analysis Benchmark [HPC09], LSQB [Mh21], gMark [Ba17], WatDiv [Al14] etc. All of these target specific use cases but without considering any OLTP. The rapid growth of social media led to the development of benchmarks modelling social networks, e.g. LinkBench [Ar13], Linked Data Benchmark Council (LDBC) Social Network Benchmark (SNB) [Er15; LDBC FinBench] etc. However, unlike GTPC, they do not consider HTAP. Moreover, they are built around social graphs while GTPC targets product graphs (see Sect. 1). The LDBC Financial Benchmark (FinBench) targets workloads for financial scenarios. It is still ongoing, with the analytical queries not yet specified. Nevertheless, it also does not consider HTAP.

---

<sup>4</sup> <https://dbgit.prakinf.tu-ilmenau.de/code/gtpc-neo4j>

None of the aforementioned graph benchmarks tackles our use case of a product-graph-based benchmark for HTAP graph databases. GTPC stresses graph systems with transactional updates and analytical queries concurrently on the same graph. Contrary to graph databases, there are HTAP benchmarks for relational databases such as CH-benCHmark [Co11] and HTAPBench [Co17]. Both aim at merging the transactional-workload-based TPC-C benchmark and the analytical-workload-based TPC-H benchmark into a hybrid benchmark. The mixed workloads are run concurrently on the same tables in the same database.

### 3 Benchmark Design

In the following, we describe the design of the GTPC benchmark, which adapts the CH-benCHmark to a property graph model and the corresponding mixed query workload.

#### 3.1 Data Model

The product graph of GTPC is based on the Labeled Property Graph Model (or property graph) [An17]. The property graph model is widely adopted as it offers a rich representation of graphs where nodes (vertices) and relationships (edges) have types or *labels* and are associated with a set of properties stored as key-value pairs. In mathematical notation, a property graph  $G$  is a tuple

$$(N, R, sd, L, l_N, l_R, D, P_N, P_R)$$

where  $N$  is a finite set of nodes,  $R \subseteq N \times N$  is a set of relationships,  $sd : R \mapsto N \times N$  is a relationship function that maps each relationship to its source and destination nodes,  $L$  is a set of *labels* that define different types of nodes and relationships,  $l_N : N \mapsto L$  and  $l_R : R \mapsto L$  are labelling functions that assign types to nodes and relationships,  $D = \cup_i D_i$  is the union of atomic domains  $D_i$  (since nodes and relationships may have an arbitrary number of properties), and  $P_N$  and  $P_R$  are sets of node and relationship properties respectively. A node property  $p_i \in P_N$  is a partial function  $p_i : N \mapsto D_i \cup \{NULL\}$ , which assigns a property value from a domain  $D_i \in D$  to a node  $n \in N$  if  $n$  has the property  $p_i$ , otherwise  $p_i(n)$  returns *NULL*. Similarly, a relationship property  $p_j \in P_R$  is a partial function  $p_j : R \mapsto D_j \cup \{NULL\}$ , which assigns a property value from a domain  $D_j \in D$  to a relationship  $r \in R$  if  $r$  has the property  $p_j$ , otherwise  $p_j(r)$  returns *NULL*.

#### 3.2 Schema

The product graph of GTPC results from a merging of the TPC-C and TPC-H benchmarks, and their adaptation for graphs. Merging of TPC-C and TPC-H schemas had been done prior with the CH-benCHmark [Co11], a mixed workload benchmark for relational databases. We adopt some of its design considerations in GTPC.

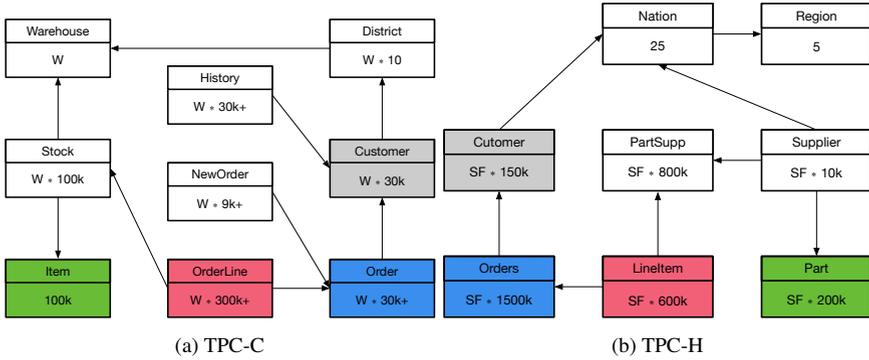


Fig. 1: (a) TPC-C Schema (b) TPC-H Schema.

**TPC-C.** The TPC-C simulates a general complex OLTP environment by way of five read-only and update-intensive transactions for the management, sale and distribution of a product or service. Specifically, it models a wholesale supplier having a certain number of warehouses and respective sales districts. The TPC-C schema covers nine relations as depicted in Fig. 1a, each associated with an entity of the database: WAREHOUSE, DISTRICT, CUSTOMER, ORDER, ORDER-LINE, ITEM, STOCK, NEW-ORDER and HISTORY. The transactions are New-Order, Payment, Order-Status, Delivery, and Stock-Level. In Fig. 1a, the arrows point in the direction of many-to-one relationships between the tables. The numbers denote the table cardinalities (the number of rows) and are expressed as factors of W, the number of warehouses, to show the scaling of the database. The “+” sign means that the number is subject to small variations as the number of rows changes [TPC-C10].

**TPC-H.** The TPC-H on the other hand simulates a business analytics application. It consists of 22 analytical queries for operations such as pricing, shipping management and market study. The business queries are executed on 8 relations: REGION, NATION, SUPPLIER, CUSTOMER, ORDER, LINEITEM, PART and PARTSUPP, as shown in Fig. 1b. Similarly to Fig. 1a, the arrows in Fig. 1b point in the direction of many-to-one table relationships and the numbers represent the cardinalities of the tables, expressed as factors of SF, the Scale Factor, which determines the database size [TPC-H21]. As we are adopting aspects of the CH-benchmark, we use TPC-H instead of TPC-DS [TPC-DS21]. Moreover, unlike TPC-H, TPC-DS has a snowflake schema with multiple dimension and fact tables. It also incorporates an Extract-Transform-Load (ETL) process, which is contrary to HTAP systems. And with regards to having a unified schema for the mixed workloads, the similarities between the TPC-C schema and the TPC-H schema make TPC-H more suitable.

**GTPC.** GTPC combines the entities of TPC-C and TPC-H, where the entities are transformed into nodes. Both TPC-C and TPC-H share CUSTOMER and ORDER as common

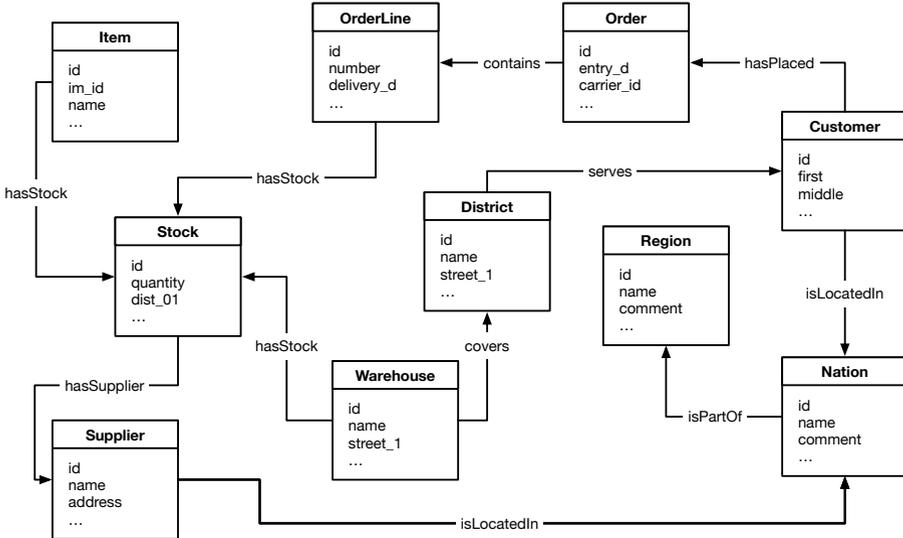


Fig. 2: GTPC Schema.

entities. Also, `ORDERLINE` and `ITEM` entities of TPC-C map to `LINEITEM` and `PART` entities of TPC-H respectively. We thus maintain the entities of TPC-C as base entities and incorporate the remaining `SUPPLIER`, `REGION` and `NATION` entities of TPC-H which are used solely for the analytical queries. By incorporating `NATION`, there is a need to introduce a relationship function  $sd$  that associates customers to their respective nations. Similarly to the `CH-benCHmark`, we take it from the first character of the `state` property of a `CUSTOMER`, which has 62 distinct values. Thus, we top up the 25 nations of the TPC-H to 62 nations.

We introduce identifiers for all node entities. This is required since, unlike in the relational model where entities contain foreign key attributes to denote relationships, nodes in a property graph do not store reference properties to other nodes. Each and every node of any given *label* is associated with an identifier that is unique to it within nodes of the same label. However, the identifiers are not unique across different labels. Nevertheless, each node is identifiable irrespective of the nodes to which it is connected, obviating the typical identifiers that are composites of foreign keys in the relational model.

Five out of the eight attributes of the `HISTORY` entities in TPC-C are foreign key attributes. As the benchmark does not require the history entities to be uniquely identifiable, we simply merge the remaining three attributes into the `CUSTOMER` entity in GTPC. As a result, `HISTORY` is not a separate entity in GTPC. This is a property graph schema optimization step that we take to avoid extra relationship traversal when retrieving history information. Additionally, we save space because the history data are simply stored as property key-values instead of as entire node objects [A121]. Similarly, we do not store `NEW-ORDER` as separate node entities. And coupled with the fact that there are no attributes associated

with `NEW-ORDER` entities, we simply extend the properties of `ORDER` nodes with an extra property that indicates whether or not an `ORDER` node is new. The graph model elevates relationships to *first-class entities*. We thus model all relationships between all pairs of node entities as separate relationship entities. Fig. 2 depicts the resulting GTPC schema.

Note that *Part\_Supp* in the TPC-H schema is an instance of a `PART` entity supplied by a certain `SUPPLIER`. In the TPC-C schema however, `STOCK` is an instance of an `ITEM` entity available in a certain `WAREHOUSE`. Since `ITEM` maps to `PART`, it follows that the information in *Part\_Supp* in the TPC-H schema is analogous to that in `STOCK` in the GTPC schema. Hence, there is a relationship between `SUPPLIER` and `STOCK` in the GTPC schema (`SUPPLIER` does not exist in the TPC-C schema) much like there is a foreign key relationship between `SUPPLIER` and *Part\_Supp* entity.

Future work includes addition of more graph features in the schema, e.g. self-edges. Self-edges could be introduced via `SUPPLIER` to `SUPPLIER` edges where bigger suppliers supply to smaller ones, via merging the `REGION` and `NATION` entity types into a single entity type, thereby transforming the existing regular edges with label `isPartOf` into self-edges etc.

### 3.3 Data Generation

Product graphs in GTPC are generated as property graphs based on the GTPC schema.

**Generator:** The GTPC graph generator is adapted from the CH-benCHmark data generator [CH]. Different graph generators employ various distributions such as power-law, Zipfian, uniform etc [Bo20]. As a first step, we simply follow the original TPC specification and adapt it in terms of assigning node degrees and property values, i.e. based on a uniform distribution. The graph data sets are output as CSV files.

**Scaling:** We use the number of warehouses as the basic scaling unit, similar to TPC-C. It determines the total number of nodes and other graph characteristics such as the degrees of all nodes, with the exception of `ITEM`, `SUPPLIER`, `REGION` and `NATION`.

### 3.4 Query Workloads

GTPC stresses the execution of concurrent OLTP and OLAP graph workloads. GTPC's OLTP graph workload is an adaptation of the five TPC-C transactions while its OLAP graph workload is an adaptation of the 22 TPC-H queries. We implement the OLTP and OLAP queries in C++ using the set of operators provided by Poseidon<sup>5</sup>. For other graph systems, the queries simply need to be implemented in the corresponding query language. As an example, we show the implementation of the GTPC OLAP #4 for Neo4j in List. 1.

---

<sup>5</sup> [https://dbgit.prakinf.tu-ilmenau.de/code/poseidon\\_core](https://dbgit.prakinf.tu-ilmenau.de/code/poseidon_core)

**OLTP:** GTPC OLTP queries 1–5 are transformations of the New-Order, Payment, Order-Status, Delivery, and Stock-Level transactions respectively into graph queries.

*OLTP #1:* This read-write transaction inserts an `ORDER` node along with a number of `ORDERLINE` nodes associated with it. The association is captured by adding a `contains` relationship for each `ORDERLINE` node, with the `ORDER` and `ORDERLINE` nodes as source and destination nodes respectively. Note that no `NEW-ORDER` nodes are created. Rather, the newly created `ORDER` nodes have one of their properties set to indicate they are new orders. Also note that additional relationships are created to connect the inserted `ORDER` node with its respective `CUSTOMER` node, and to connect the inserted `ORDERLINE` nodes to their respective `STOCK` nodes. The transaction also entails other read and write operations like retrieving `WAREHOUSE`, `DISTRICT`, `CUSTOMER` and `STOCK` nodes, as well as projecting some of their properties; and updating the properties of `DISTRICT` and `STOCK` nodes.

*OLTP #2:* This transaction updates a customer’s balance. The transaction additionally updates the `WAREHOUSE` and `DISTRICT` nodes so that the sales tally with the new payment.

*OLTP #3:* This read-only transaction checks the order status of a customer. It covers retrieval of the `CUSTOMER` node, a relationship traversal(s) to retrieve the `ORDER` node(s), and a further relationship traversal(s) to retrieve the `ORDERLINE` node(s) connected to it.

*OLTP #4:* This read-write transaction delivers a batch of 10 new orders. It consists in retrieving the `ORDER` node and updating its properties to signify that it has been delivered and thus no longer a new order. Besides that, in order to reflect the delivery, its relationships are traversed to retrieve and update the delivery date properties of its associated `ORDERLINE` nodes; as well as the balance and delivery count of its associated `CUSTOMER` node.

*OLTP #5:* This read-only transaction computes the number of stock items sold recently and having a stock level below a certain threshold value. The transaction entails relationship traversals of up to four levels and an aggregate operation (`count`).

**OLAP:** We adapt the 22 queries of TPC-H for graphs, resulting in GTPC’s OLAP queries 1–22. The TPC-H presents systems with a rich set of *chokepoints* or challenges with respect to optimized and efficient query processing. An analysis of the TPC-H chokepoints is presented in [BNE13]. The authors integrated chokepoints in their design of the LDBC benchmarks [LDBC SNB]. GTPC thus preserves those chokepoints. Additionally, since the OLAP queries are largely traversal operations ranging from one to eight hops, GTPC thus tests a system’s ability to efficiently traverse the graph topology by choosing the optimal traversal order. This is central to the performance of graph processing.

**Execution Mode:** GTPC’s mixed workload is run as concurrent streams of OLTP and OLAP queries. We dispatch an OLTP stream as a randomly permuted yet complete set of OLTP queries while an OLAP stream consists of the full set of OLAP queries ordered sequentially. Each OLAP stream starts with a different query. Each stream (OLTP or OLAP) is assigned a thread from a thread pool and, within the thread, the queries of the stream are

executed sequentially. Hence, the mixed execution mode consists in a mix of OLTP and OLAP streams that are dispatched in parallel from the thread pool and executed concurrently. As part of future work, an aspect of the execution to consider for fair comparison, especially as to systems with high OLTP performance, is bounding the graph size. This could be done by converting inserts into updates after a certain size limit or by using the number of OLTP streams that results in the maximum OLTP throughput.

```
MATCH(o:Order)-[:contains]->(ol:OrderLine)
WHERE o.entry_d >= datetime('2007-01-02T00:00:00.000000')
      AND o.entry_d < datetime('2012-01-02T00:00:00.000000')
      AND ol.delivery_d >= o.entry_d
WITH o.ol_cnt AS o_ol_cnt, COUNT(*) AS order_count
RETURN o_ol_cnt, order_count
```

List. 1: GTPC OLAP #4 in Cypher.

### 3.5 Benchmark Parameters

Our benchmark parameters are the database size in terms of the total number of nodes, which is a function of the number of warehouses; the number of concurrent OLTP and OLAP streams, which determines the level of contention between transactions; and the transaction isolation level, which determines transactional guarantees.

### 3.6 Performance Metrics

Most of the benchmarks discussed make use of execution time (latency) and/or throughput as performance metrics. Other metrics considered in benchmarking include CPU usage, memory footprint etc. In GTPC, we currently evaluate OLTP-OLAP performance interplay based on execution time and throughput.

## 4 Evaluation

We use our graph database, Poseidon [Ji21], as an example graph system for our evaluation in this paper. It should be noted here that although Poseidon is based on persistent memory, however, persistent memory is not relevant to GTPC. Poseidon is only a graph system we use here to make an example implementation of the GTPC benchmark. For concurrency control in this evaluation, we use the multi-version two-phase locking protocol, where the number of versions is limited to two (2V2PL).

We conduct our evaluations on a dual-socket Intel Xeon Gold 5215 with 10 cores per socket running at a maximum of 3.40 GHz. The machine is equipped with 384 GB DRAM, 1.5

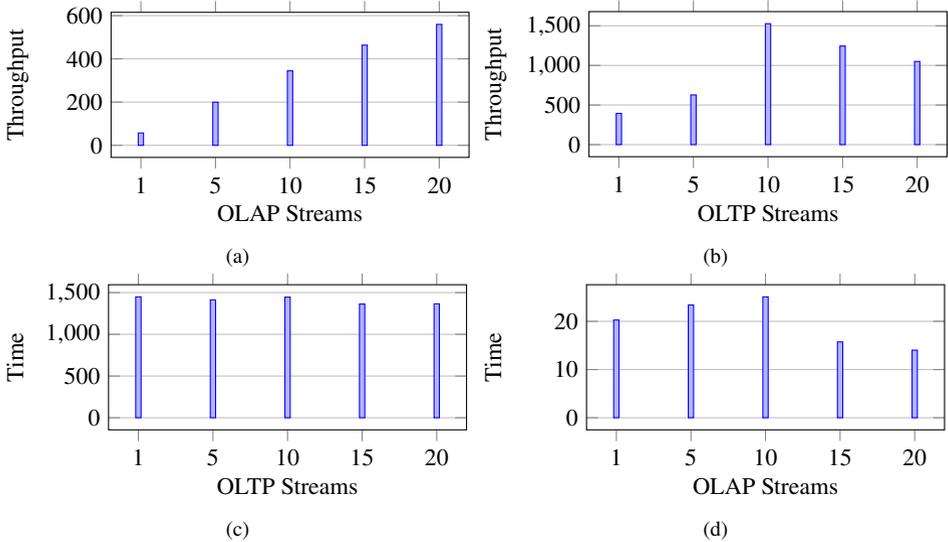


Fig. 3: (a) OLAP-only throughput (in Qph) with increasing number of OLAP streams (b) OLTP-only throughput (in Qph) with increasing number of OLTP streams (c) Execution time of OLTP stream (in sec) with increasing number of OLTP streams (d) Execution time of OLTP stream (in sec) with increasing number of OLAP streams.

TB Intel Optane DC Persistent Memory Module (DCPMM) operating in AppDirect mode, 4x 1.0 TB Intel SSD DC P4501 Series connected via PCIe 3.1; and runs on CentOS 7.9 with Linux Kernel 5.10.6. We use the Intel Persistent Memory Development Kit (PMDK) version 1.9.1 and libpmemobj-cpp version 1.11 for directly accessing the PMem device. Meanwhile, all executions were fixed to a single socket to factor out NUMA effects.

We load the GTPC dataset of two warehouses into Poseidon to execute the GTPC workloads. The input graph has 1,031,312 nodes and 1,992,528 relationships. We first execute OLAP streams exclusively. There is thus no contention for graph objects in this setting. Although the latency per OLAP stream increases with an increasing number of concurrent streams, Fig. 3a shows that the overall query throughput (expressed in queries per hour) increases. With OLTP-only stream execution, however, contention between transactions on graph objects results in some of them aborting. Also, different transactions abort at different stages of execution – with computation being wasted for each transaction that aborts. All these depend on the graph characteristics and workload pattern. More skew would result in higher contention on common graph objects, as transactions are more likely to access the same nodes – especially those with higher degrees. We currently adopt the TPC specification in our relationship mapping and substitution parameters. Nevertheless, we see in Fig. 3b that transaction throughput increases initially with an increase in concurrent OLTP streams until

it reaches a threshold at 10 streams, after which dispatching additional concurrent streams decreases throughput.

Thereafter, we run the mixed OLTP-OLAP workload to demonstrate the effectiveness of GTPC in testing performance isolation in a system – i.e. the system’s ability to handle the interference between concurrently running OLTP and OLAP query streams without the performance of either of the two being compromised as a result of the other. We start by fixing the OLAP stream at a single stream while varying the concurrent OLTP streams. Fig. 3c shows the execution times (in seconds) of the individual queries in the OLAP stream when run concurrently with a varying number of OLTP streams. For each query, its execution times in the presence of the varying number of OLTP streams lie within a relatively narrow range. This shows that the OLTP streams do not interfere much with the OLAP query execution times, as the OLTP transactions do not block the OLAP queries due to multi-versioning. Compared with the earlier OLAP-only runs, we see that the execution time of an OLAP stream is influenced more by concurrent OLAP streams than by the same number of OLTP streams. Finally, we maintain a single OLTP stream and run it concurrently with a varying number of OLAP streams. We expect the OLTP stream execution time to increase with more OLAP streams partly because the improved concurrency of OLAP queries in the 2V2PL is a trade-off with delaying transaction commit when the OLTP transactions wait for OLAP queries before acquiring a *certify lock*. The initial part of Fig. 3d shows that. We note here that we omit the execution time of OLTP #4, which is the most write-heavy transaction after OLTP #1, as it fails starting from 10 OLAP streams.

## 5 Conclusion

In this paper, we have presented our ongoing work on GTPC, a hybrid OLTP-OLAP graph benchmark based on the TPC-C and TPC-H benchmarks. We implemented and ran GTPC on Poseidon, our graph database, as an example to showcase the effectiveness of GTPC in testing HTAP graph systems with respect to performance isolation between concurrently running OLTP and OLAP graph query streams. Future work towards developing GTPC into a fully-fledged hybrid graph benchmark include extending the mixed workload to further include graph algorithms, incorporating more real-world data characteristics in the data generation, introducing more benchmark parameters and performance metrics to facilitate better comparison between different HTAP graph systems.

**Acknowledgements.** This work was partially funded by the German Research Foundation (DFG) in the context of the project “Hybrid Transactional/Analytical Graph Processing in Modern Memory Hierarchies (#TAG)” (SA 782/28-2) as part of the priority program “Scalable Data Management for Future Hardware” (SPP 2037) and by the Carl-Zeiss-Stiftung under the project “Memristive Materials for Neuromorphic Electronics (MemWerk)”.

## References

- [Al14] Aluç, G.; Hartig, O.; Özsu, M. T.; Daudjee, K.: Diversified Stress Testing of RDF Data Management Systems. In (Mika, P.; Tudorache, T.; Bernstein, A.; Welty, C.; Knoblock, C. A.; Vrandečić, D.; Groth, P.; Noy, N. F.; Janowicz, K.; Goble, C. A., eds.): *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference*, Riva del Garda, Italy, October 19-23, 2014. *Proceedings, Part I*. Vol. 8796. *Lecture Notes in Computer Science*, Springer, pp. 197–212, 2014, URL: [https://doi.org/10.1007/978-3-319-11964-9%5C\\_13](https://doi.org/10.1007/978-3-319-11964-9%5C_13).
- [Al21] Alotaibi, R.; Lei, C.; Quamar, A.; Efthymiou, V.; Özcan, F.: Property Graph Schema Optimization for Domain-Specific Knowledge Graphs. In: *37th IEEE International Conference on Data Engineering, ICDE 2021*, Chania, Greece, April 19-22, 2021. IEEE, pp. 924–935, 2021, URL: <https://doi.org/10.1109/ICDE51399.2021.00085>.
- [An17] Angles, R.; Arenas, M.; Barceló, P.; Hogan, A.; Reutter, J. L.; Vrgoč, D.: *Foundations of Modern Query Languages for Graph Databases*. *ACM Comput. Surv.* 50/5, 68:1–68:40, 2017, URL: <https://doi.org/10.1145/3104031>.
- [Ar13] Armstrong, T. G.; Ponnepkanti, V.; Borthakur, D.; Callaghan, M.: LinkBench: a database benchmark based on the Facebook social graph. In (Ross, K. A.; Srivastava, D.; Papadias, D., eds.): *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, New York, NY, USA, June 22-27, 2013. ACM, pp. 1185–1196, 2013, URL: <https://doi.org/10.1145/2463676.2465296>.
- [Ba17] Bagan, G.; Bonifati, A.; Ciucanu, R.; Fletcher, G. H. L.; Lemay, A.; Advokaat, N.: gMark: Schema-Driven Generation of Graphs and Queries. *IEEE Trans. Knowl. Data Eng.* 29/4, pp. 856–869, 2017, URL: <https://doi.org/10.1109/TKDE.2016.2633993>.
- [Be19] Besta, M.; Peter, E.; Gerstenberger, R.; Fischer, M.; Podstawski, M.; Barthels, C.; Alonso, G.; Hoefler, T.: Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *CoRR abs/1910.09017*, 2019, arXiv: 1910.09017, URL: <http://arxiv.org/abs/1910.09017>.
- [BNE13] Boncz, P. A.; Neumann, T.; Erling, O.: TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In (Nambiar, R.; Poess, M., eds.): *Performance Characterization and Benchmarking - 5th TPC Technology Conference, TPCTC 2013*, Trento, Italy, August 26, 2013, *Revised Selected Papers*. Vol. 8391. *Lecture Notes in Computer Science*, Springer, pp. 61–76, 2013, URL: [https://doi.org/10.1007/978-3-319-04936-6%5C\\_5](https://doi.org/10.1007/978-3-319-04936-6%5C_5).

- [Bo20] Bonifati, A.; Holubová, I.; Prat-Pérez, A.; Sakr, S.: Graph Generators: State of the Art and Open Challenges. *ACM Comput. Surv.* 53/2, 36:1–36:30, 2020, URL: <https://doi.org/10.1145/3379445>.
- [CH] CH-benCHmark, URL: <https://db.in.tum.de/research/projects/CHbenCHmark/index.shtml>.
- [Co11] Cole, R. L.; Funke, F.; Giakoumakis, L.; Guy, W.; Kemper, A.; Krompass, S.; Kuno, H. A.; Nambiar, R. O.; Neumann, T.; Poess, M.; Sattler, K.; Seibold, M.; Simon, E.; Waas, F.: The mixed workload CH-benCHmark. In (Graefe, G.; Salem, K., eds.): *Proceedings of the Fourth International Workshop on Testing Database Systems, DBTest 2011, Athens, Greece, June 13, 2011*. ACM, p. 8, 2011, URL: <https://doi.org/10.1145/1988842.1988850>.
- [Co17] Coelho, F.; Paulo, J.; Vilaça, R.; Pereira, J.; Oliveira, R.: HTAP-Bench: Hybrid Transactional and Analytical Processing Benchmark. In (Binder, W.; Cortellessa, V.; Koziolk, A.; Smirni, E.; Poess, M., eds.): *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*. ACM, pp. 293–304, 2017, URL: <https://doi.org/10.1145/3030207.3030228>.
- [Er15] Erling, O.; Averbuch, A.; Larriba-Pey, J. L.; Chafi, H.; Gubichev, A.; Prat-Pérez, A.; Pham, M.; Boncz, P. A.: The LDBC Social Network Benchmark: Interactive Workload. In (Sellis, T. K.; Davidson, S. B.; Ives, Z. G., eds.): *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM, pp. 619–630, 2015, URL: <https://doi.org/10.1145/2723372.2742786>.
- [HPC09] HPC Scalable Graph Analysis Benchmark, 2009, URL: <http://www.graphanalysis.org/benchmark/GraphAnalysisBenchmark-v1.0.pdf>.
- [Ji21] Jibril, M. A.; Baumstark, A.; Götze, P.; Sattler, K.: JIT happens: Transactional Graph Processing in Persistent Memory meets Just-In-Time Compilation. In (Velegarakis, Y.; Zeinalipour-Yazti, D.; Chrysanthis, P. K.; Guerra, F., eds.): *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. OpenProceedings.org, pp. 37–48, 2021, URL: <https://doi.org/10.5441/002/edbt.2021.05>.
- [LDBC FinBench] The LDBC Financial Benchmark, URL: [https://ldbcouncil.org/ldbc\\_finbench\\_docs/ldbc-finbench-specification.pdf](https://ldbcouncil.org/ldbc_finbench_docs/ldbc-finbench-specification.pdf).
- [LDBC SNB] The LDBC Social Network Benchmark, URL: [http://ldbc.github.io/ldbc\\_snb\\_docs/ldbc-snb-specification.pdf](http://ldbc.github.io/ldbc_snb_docs/ldbc-snb-specification.pdf).

- [Mh21] Mhedhbi, A.; Lissandrini, M.; Kuiper, L.; Waudby, J.; Szárnyas, G.: LSQB: a large-scale subgraph query benchmark. In (Kalavri, V.; Yakovets, N., eds.): GRADES-NDA '21: Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), Virtual Event, China, 20 June 2021. ACM, 8:1–8:11, 2021, URL: <https://doi.org/10.1145/3461837.3464516>.
- [Neo4j] Neo4j, URL: <https://neo4j.com/>.
- [Sa20] Sahu, S.; Mhedhbi, A.; Salihoglu, S.; Lin, J.; Özsü, M. T.: The ubiquity of large graphs and surprising challenges of graph processing: extended survey. VLDB J. 29/2-3, pp. 595–618, 2020, URL: <https://doi.org/10.1007/s00778-019-00548-x>.
- [TigerGraph] TigerGraph, URL: <https://www.tigergraph.com>.
- [TPC-C10] TPC-C Specification, 2010, URL: [http://tpc.org/tpc\\_documents\\_current\\_versions/pdf/tpc-c\\_v5.11.0.pdf](http://tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf).
- [TPC-DS21] TPC-DS specification, 2021, URL: [https://www.tpc.org/tpc\\_documents\\_current\\_versions/pdf/tpc-ds\\_v3.2.0.pdf](https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf).
- [TPC-H21] TPC-H Specification, 2021, URL: [http://tpc.org/tpc\\_documents\\_current\\_versions/pdf/tpc-h\\_v3.0.0.pdf](http://tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.0.pdf).
- [Ultipa] Ultipa, URL: <https://www.ultipa.com/>.
- [Zh20] Zhu, X.; Serafini, M.; Ma, X.; Aboulnaga, A.; Chen, W.; Feng, G.: Live-Graph: A Transactional Graph Storage System with Purely Sequential Adjacency List Scans. Proc. VLDB Endow. 13/7, pp. 1020–1034, 2020, URL: <http://www.vldb.org/pvldb/vol13/p1020-zhu.pdf>.



# IBM Data Gate: Making On-Premises Mainframe Databases Available to Cloud Applications

Knut Stolze,<sup>1</sup> Felix Beier,<sup>1</sup> Vassil Dimov,<sup>1</sup> Eirini Kalogeiton,<sup>1</sup> Mateo Tošić<sup>1</sup>

## Abstract:

Many companies use databases on the mainframe for their mission critical applications. It is important to exploit this existing data for analysis and business decisions via modern applications that are often built exclusively for cloud environments. IBM Db2 for z/OS Data Gate (Data Gate) is bridging the gap between mainframe databases and such cloud-native applications. It offers high-performance data synchronization for connecting both worlds, while providing data coherence at the level of individual transactions. Data Gate is a hybrid cloud solution that protects existing systems, applications, and investments into those, while enabling new use cases to work with mainframe data. In this paper, we give an overview of Data Gate and discuss how it evolved from IBM Db2 Analytics Accelerator (IDAA) technology by adjusting the system architecture and some of the functionality in order to make IBM Db2 for z/OS (Db2/z) data a first-class citizen the cloud.

## 1 Introduction

Database systems are the well-established approach to manage application data since the relational model was invented. In 1983, IBM released Db2/z, a relational DBMS for IBM's *mainframe*. As of today, Db2 and Db2/z are the core data management products for many organizations, as they ensure excellent availability, performance, scalability, and storage saving options. Nowadays, the need of organizations has grown to take advantage of their huge amount of data that is generated or collected every day. To cope with this need, cloud solutions are introduced, providing a simple, efficient, cost-effective, scalable, and flexible environment [Ch15]. IBM offers various solutions for making on-premises data sources, such as Db2/z, available as first-class citizens in cloud-native environments while addressing a series of requirements, like ensuring that business-critical systems are not impacted negatively, existing investments are protected, and risks to such systems are minimized.

This paper focusses on an approach for *integrating* on-premises databases into the new cloud-native landscape rather than *migrating* such systems. While this integration methodology protects decades of customers' investments in their data management infrastructure, it rises a set of requirements that have to be addressed in order to seamlessly integrate business-critical databases into a managed cloud ecosystem, without having to relocate existing applications and/or workloads. First, integration solutions have to optimize data placement with caching strategies in order to collocate cloud-based data accesses. Those cloud-based caches need

---

<sup>1</sup> IBM Germany Research & Development GmbH, {stolze, febe}@de.ibm.com, {Vassil.Dimov1, Eirini.Kalogeiton, Mateo.Tosic}@ibm.com

to be synchronized with the on-premises data sources, which comes with high requirements w.r.t. change replication performance. Second, any data replication mechanism inevitably introduces latencies that need to be hidden by some cache coherency protocols to guarantee consistent views when data is accessed by consuming applications. Third, compatibility aspects need to be considered. Although the SQL standard pretends a uniform specification of relational data management and processing, the actual systems implementing them differ largely in available features and functions. These differences need to be hidden from customer applications for a truly transparent data access experience. Finally, various non-functional requirements w.r.t. security, administration, monitoring, stability, etc. have to be satisfied for enterprise readiness. In the following, we will discuss how Data Gate [IB22e] solved these challenges for making Db2/z data available and synchronized in the cloud. Based on well-proven database accelerator technologies of IDAA [IB22d], Data Gate replicates Db2/z data into a cloud database and uses patented cache coherence protocols to guarantee data consistency, irrespective from where the data is accessed. Data Gate can be considered the cloud evolution of IDAA. We will present important aspects of Data Gate's architecture and will also shed some light on architectural decisions that led to dead ends.

## 2 Related Work

Data Gate is primarily an integration component that makes data available from mainframe databases (ie. Db2/z) in established cloud databases. Generally, different approaches exist to integrate on-premise data sources in the cloud. The most common ones are described below.

The **Lift-&Shift** approach transfers not only the data, but the complete on-premises environments, consisting of data stores and the applications working with them, to a virtualized cloud-based infrastructure. This is beneficial for data stores and applications running on platforms that are available in the cloud already. *Lift-&Shift* may require recompiling code, changes to the packaging/installation procedures or security and user management due to the different underlying environment. Additionally, data and file conversions may change [Me18]. If suitable, this approach eliminates the need for on-premises environments, as it benefits from reusability and is faster than rewriting existing applications [In20]. On the other hand, future enhancements are slower to adopt because the software stack is not modernized [PP20].

**Custom data ingestion pipelines** use a combination of tools for bulk loading and continuous data replication to copy the data from on-premises databases to a cache in the cloud for minimizing data access costs of cloud-native applications. The required tools can be implemented from scratch or by combining open-source technologies, like Kafka [Kr11], Debezium [Co22] and others [De21]. Alternatively, a combination of existing products with different performance characteristics and consistency semantics can be used, e. g., AWS Database Migration Service [Ge22]. Such data ingestion pipelines minimize risks for existing applications because the source database systems are not impacted and allow to embed data transformations that may be required by consuming cloud applications. On the other hand, this approach has a longer time-to-market because multiple components are involved that have to be separately implemented and integration-tested as a whole system.

Many companies offer **integrated data replication tools** that provide all benefits of custom data ingestion pipelines while abstracting the implementation complexity, e. g. Oracle Golden Gate [Gu16] and Cyniti Data Replication [Sy19]. Those tools replicate data from the on-premises data store to a target database in the cloud, offering features for initial bulk data ingestion, continuous data replication, and utilities to orchestrate the whole process. These tools take care of encryption and data transformations and, usually, offer better data movement performance and superior consistency semantics than home-grown solutions. An integration with other cloud services is provided to ease data consumption in cloud environments. Examples for such integrations are cataloging data in enterprise data catalogs, automated data governance, and data profiling. Data Gate falls into this group.

While the previous approaches involve movement of data from on-premises data sources to the cloud, **data virtualization** allows consuming the data from cloud-based applications without creating a cached data copy. Inconsistencies between different data versions are avoided because all data consumers operate on the same data. To achieve that, a data virtualization layer is defined that combines separate data sources from different platforms. The ease of integrating various data platforms at one place enables rapid prototyping of new cloud applications [MAT19]. On the other hand, the on-premises data store has to facilitate the whole workload, analytical and transactional, stemming from both, on-premises and cloud applications. Thus, a virtualization approach can be more costly in terms of operations of the source data store and pose a risk for existing business-critical applications.

### 3 Evolution from IDAA to Data Gate

Data Gate can be categorized as integrated data replication tool that facilitates the movement of data from an on-premises mainframe database to a cloud database. Data Gate is integrated into IBM Cloud Pak for Data (CP4D) [IB22c], IBM's cloud platform for data analytics with a unified service architecture for user management, encryption, and common user experience for working with a wide variety of data sources. It interacts with the platform's metadata catalog for making mainframe data discoverable and consumable by any cloud application. Applications can access the data in the cloud database at any time with the guarantee that they always operate on the latest, consistent view of the data. Furthermore, Data Gate supports seamless routing of analytic workload stemming from Db2/z to the cloud. It evolved from IDAA, an on-premises accelerator appliance for processing analytical Db2/z queries. The architectural evolution is illustrated in Figure 1.

The architecture of IDAA that acts as the base for Data Gate is illustrated in Figure 1A. IDAA is a pre-configured cluster that runs a tuned IBM Db2 Warehouse (Db2 WH) installation and an accelerator server middleware. IDAA is deeply integrated into Db2/z as internal resource for processing analytic queries (green flow in Figure 1A). IDAA exists only as an extension to Db2/z and the data residing on the accelerator can only be accessed through Db2/z. IDAA's main purpose is to make mainframe workload more predictable and to execute the analytical part of it more efficiently. Queries that are submitted by client applications are analyzed by the Db2/z optimizer which decides to route the query to IDAA if it is

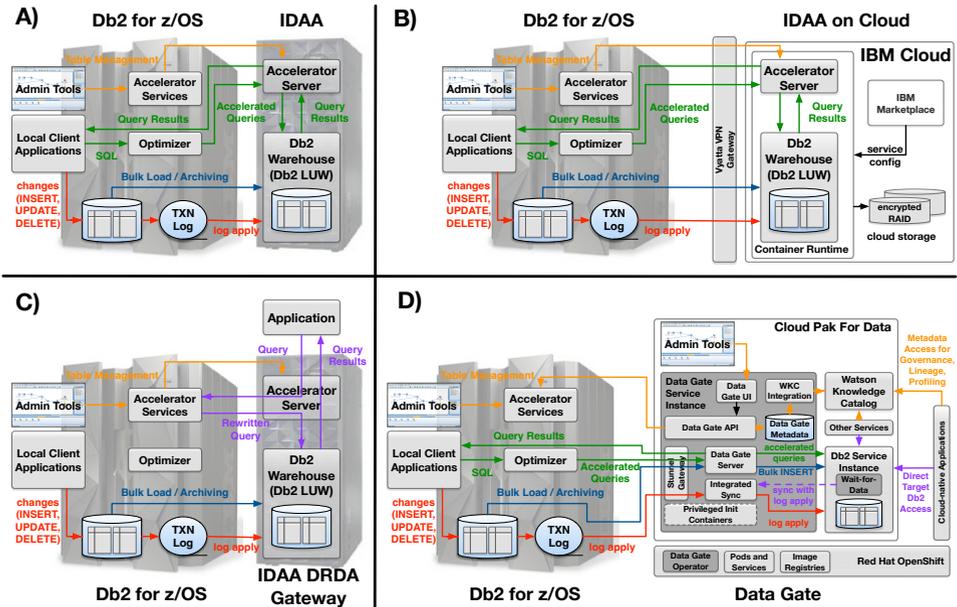


Fig. 1: IDAA and Data Gate Architecture Overview

an analytical one. Otherwise, it is executed locally. In case it is routed to IDAA, it will be transformed to the Db2 WH SQL dialect, executed on the appliance, and its result set is streamed back to the application via Db2/z. The offloading is, therefore, completely transparent to the application and no modifications are needed. That is a critical requirement for many Db2/z users. The data residing on IDAA is always kept in the same encoding as on the source database so that queries will deliver the same results as if they are run on Db2/z.

Accelerated queries are executed on a copy of the Db2/z data. An administrator selects tables that should be accelerated and adds them to the appliance (orange flow in Figure 1A), which registers the table’s schema. The actual copy of the table data is triggered via a bulk load mechanism that unloads table partitions from Db2/z in an efficient way using a dedicated utility (see [IB22b]). The partition data is transferred to the accelerator and inserted in parallel to Db2 WH (blue flow Figure 1A). The load process creates a snapshot of the tables that can later be updated with another bulk load in case many changes accumulated before the accelerator data copy needs to be refreshed.

Consistency of the copied data in the accelerator is realized by a multi-version concurrency control (MVCC) mechanism that employs views to define visible vs. invisible rows [SBM19]. Therefore, each bulk load assigns a unique ID to all processed rows which is stored in an internal column of the target table. This ID is used by a view for filtering rows when queries are executed. The view is updated after each load operation and synchronized with potential concurrent incremental update operations. Cross-table consistency is guaranteed because DDL statements are fully transactional in Db2 WH and adhere to the ACID properties.

An alternative to bulk loading is the incremental update strategy, called IBM Integrated Synchronization (InSync) (red flow in Figure 1A). It can be used if changes happen frequently and accelerated queries should run on the latest data version. InSync captures changes to observed tables from the Db2/z transaction log and applies them to the target database [Bu20]. The overall data flow is highly optimized and avoids data transformations, e. g., the raw log format written by Db2/z is directly consumed by Db2 WH. The specialized incremental update implementation is further exploited during query processing. An application can request to process the data as it was in Db2/z at the time when a query was submitted. Upon such a request, the accelerator captures the current head-of-transaction-log from Db2/z or the newest log position for the tables specified in the query. Query execution is delayed until this log position has been applied by InSync to the target database.

IDAA provides a High Performance Storage Saver (HPSS) feature to move data from Db2/z to Db2 WH. Historical partitions that do not change anymore on Db2/z can be loaded to IDAA and are removed from the source database. Only partitions that are still being updated remain on Db2/z. Although the “archived” data is not present on Db2/z anymore, it can still be queried via the accelerator [IB22a]. HPSS reduces the data volume on Db2/z which leads to faster index maintenance, more efficient reorganizations and statistics collection, and, thus, processing overall.

Another feature allows tables to exist only on the accelerator without being present on Db2/z at all [BSM16]. For those Accelerator-only Tables (AOTs), not only analytical queries but also all Data Manipulation Language (DML) statements are routed to the accelerator. AOTs can be used for efficient in-database transformations, data preparation tasks, and are a perfect fit for storing temporary data for subsequent queries or reports.

IDAA’s architecture has been adjusted several times in the past in order to enable additional use cases and data access patterns. Figure 1B) illustrates the IDAA on Cloud modification that provided query acceleration capabilities as standalone cloud service [BS17]. The data flow and use cases are comparable to the appliance form factor of IDAA. From a high-level perspective, just the deployment of the accelerator software stack changed so that it can be hosted in IBM’s public and private cloud environments. Therefore, the Db2 WH as well as the accelerator middleware have been containerized. To meet the security requirements of a cloud-based database service offering, encryption mechanisms were introduced, which were not needed before in the isolated on-premises environment of IDAA. Both, data at rest, i. e. the table copies stored inside Db2 WH, as well as data in motion, i. e. data that flows between Db2/z and the accelerator, are protected via encrypted storage and VPN gateways.

A second direction of the IDAA architecture aimed at opening the encapsulated target database to minimize data transfer overheads for large result sets to external applications. The corresponding IDAA DRDA Gateway architecture is depicted in Figure 1C). An application can directly connect to the accelerator for executing a query via the DRDA protocol [DRD03]<sup>2</sup> (purple flow in Figure 1C). From there, the query is passed on to Db2/z

---

<sup>2</sup> For IBM’s database systems, the DRDA protocol is the underlying technology used by ODBC/CLI or JDBC.

where the statement is validated (privileges, object existence, etc.) and rewritten according to the SQL dialect of Db2 WH. Db2/z routes the query to the accelerator, which hands it over to the cloud database for execution. Contrary to normal query execution, IDAA returns an empty result set to Db2/z while forwarding the actual result set from Db2 WH to the application.

Although the DRDA gateway showed a 10x performance improvement for transferring result sets, data could not be directly accessed. All queries had to be processed by Db2/z during preparation and IDAA was handling the execution in Db2 WH. IDAA itself was an additional hop for transferring result sets and applications had to use the SQL constructs of Db2/z and could not take advantage of native constructs available in the cloud database. Hence, the product never got beyond the prototyping stage. The standalone IDAA on Cloud service was also discontinued because it lacked a deep integration into a cloud-based data analytics environment.

A third direction was the integration of additional data processing engines with the target database. For example, IBM Netezza Analytics Stored Procedures [IB16, BSM16] have been made available as separate package that could be added to IDAA. The package enabled additional stored procedures for analytics that directly run in the target database. The stored procedures were callable in Db2/z, but processing was forwarded to the accelerator. The intention was to generalize the query acceleration idea to custom analytical packages. Spark was also provided as additional processing engine that was collocated with the target database, Netezza at that point in time. Stored procedures could be used to submit Spark jobs. However, a shift in the underlying database technology from Netezza to Db2 WH lead to a setback.

The path of Db2/z data towards the cloud has been paved by Data Gate which can generally be regarded as consolidation of the previous architectures. Data Gate's architecture is illustrated in Figure 1D), which will be explained in more details in Section 4.

## **4 Deep Dive into Data Gate**

### **4.1 Data Gate Architecture**

Data Gate mainly inherited its functionality from IDAA on Cloud. IDAA and Data Gate can coexist and be connected to the same Db2/z system. While IDAA is attached locally and used for accelerating analytic queries, Data Gate is an extension of Db2/z to a cloud-based environment where it maintains a cached twin copy of the tables. In contrast to IDAA, Data Gate allows direct access from applications to the target database. In fact, it is the user's responsibility to provide, configure, and maintain a Db2 on Cloud instance and connect it to Db2/z via Data Gate. This offers the flexibility to use the cached Db2/z data for various use cases in the cloud-based environment.

Unlike IDAA on Cloud, Data Gate is integrated in a common cloud architecture (cf. section 4.2), which is more flexible since it allows organizations to tailor their system architecture towards specific needs of the target applications by allowing:

- Independent configuration of the computation and storage resources that should be allocated for Data Gate and the cloud database instance
- Selection between multiple storage types with different capabilities with regard to failover and performance characteristics
- Selection of different cloud database form factors: row store for OLTP workloads, Db2 WH for analytics, and Db2 WH with query acceleration for OLAP workloads from cloud-native applications and accelerated query routing via Db2/z, like IDAA

Internally, Data Gate uses a microservices architecture. Therefore, IDAA's monolithic middleware was split into more fine-granular component containers which are loosely coupled and controlled over an API layer that maintains stateful information and metadata. These APIs communicate with Db2/z over secure connections and invoke administrative stored procedures that are used to control IDAA or Data Gate. Data Gate uses encryption in all layers of communication and data propagation (cf. section 4.2). This differs from IDAA where a dedicated, private network is used to avoid any encryption-related overhead.

## 4.2 Cloud Platform Mandates

Data Gate is available through CP4D that is deployed on top of an Red Hat OpenShift (OpenShift) cluster. One advantage of OpenShift is the use of operators for integrating all components. A Data Gate operator observes container registries and automatically reconciles the cluster on updates, e. g., for security patches, which can be applied in short time intervals. Moreover, CP4D provides a set of common services that can be used by any service offering, like user and credentials management, logging and diagnostics, and a common user interface. Unlike IDAA, Data Gate enables direct access to the cloud database, which allows to consume Db2/z data by cloud-native applications. Since the target database is not fully controlled by Data Gate, high security requirements are implemented by:

- Providing fewer privileges for run-time users
- Deploying a dedicated init container for separation of duties and target database tuning
- Encrypting data everywhere: at rest and in motion between all components, e. g., TLS encryption between Db2/z and Data Gate is provided via a dedicated *stunnel* container

## 4.3 Integration with Cloud Services

Most organizations have huge amounts of data stored in many forms in various locations. Finding relevant data quickly and connecting disparate data sources can be challenging and time-consuming. IBM Watson Knowledge Catalog (WKC) [IB22f] unites all information assets into a single metadata catalog. A single click in Data Gate's User Interface (UI) is sufficient to publish metadata about the source database connection and its data assets (tables), along with the target database connection and all replicated data assets to WKC (orange data/metadata flow in Figure 1D). In addition to making the metadata discoverable, the main purpose of the integration is to enable WKC's built-in tools for data governance, lineage, and profiling, as well as numerous other cloud-native applications, e. g., Watson Studio for data analysis. This is particularly useful to ensure that each user can only see data according to their roles, where WKC masks and randomizes sensitive data.

#### 4.4 Changes in the Backend Database

The target database is not owned by Data Gate but integrated as separate cloud service. To ensure high performance of the whole system the target database has to be tuned. For example, the archive log is disabled for better data synchronization performance in any deployment. Other parameters, such as automated statistics maintenance and table reorganizations, are enabled for the analytical use case only. For protecting the runtime environment, the tuning is done in a special init container that runs as privileged user in comparison to the normal Data Gate operations (see Figure 1D).

Data Gate caches data in the cloud for new applications which require data in UNICODE rather than EBCDIC encoding that is typically used in Db2/z. Thus, Data Gate is re-encoding the data when it is copied. Because such code page conversions may increase string lengths, the column widths of target tables is increased based on heuristics that implement a trade-off between the range of supported values, maximum column widths, and storage utilization.

In IDAA, queries are routed from Db2/z to the target database. With Data Gate, the cloud database is directly accessible by applications. The replicated database is impacted by the data synchronization latency and is only eventually consistent to the source database – similarly to most asynchronously replicated databases. Eventually consistent systems make no guarantees for the staleness of the data [Vo09]. Applications accessing the target database could retrieve data older than the one already persisted in the source database. Similarly to IDAA, Data Gate provides a way to run queries on the target database with the newest data from Db2/z. Query can retrieve transactionally consistent data as if the execution happens on the source database by using new SQL syntax in the cloud database, which uses Data Gate under the covers. A query can be annotated to *wait* for the newest data from the on-premises database to be replicated to the cloud database:

```
SET CURRENT QUERY WAITFORDATA = 10;  
SELECT COUNT(*) FROM BANK.TRANSACTION;
```

The first line sets the `WAITFORDATA` special register of the cloud database to 10 seconds. The subsequent query will wait until the most recent changes from the source database have been replicated or the 10 seconds timeout has expired. Since Data Gate comes with very low latency (usually a few seconds only), query execution commences well before the timeout expires. If the timeout is reached, an appropriate error is returned. With this extension, *read-after-write inconsistencies* [Je] are avoided for applications that access data on the target database.

## 5 Performance Evaluation

After discussing the most important architectural changes that transform IDAA from a highly tuned on-premises analytical database accelerator to Data Gate as cloud database replication tool, this section sheds some light on the performance impact of these changes. From a use case perspective, we compared the performance of the initial data loading

phase (bulk load), the incremental update performance for replicating changes from the source database to the target database, and also the query acceleration flow from Db2/z. Additional Data Gate use cases that employ different Db2 service versions as target database or different data access patterns, such as direct query processing from cloud applications on the target database, are out of scope for this paper.

| Operation                        | IDAA  | Data Gate |
|----------------------------------|-------|-----------|
| Initial Load Throughput (TB/h)   | 4.2   | 1.4       |
| Average InSync Throughput (Tx/s) | 511 k | 364 k     |
| Average InSync Latency (s)       | 6.4   | 10.4      |
| Total Query Runtime (s)          | 969   | 1260      |

Tab. 1: Operations

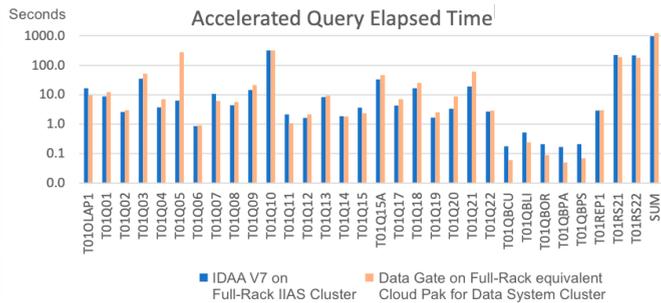


Fig. 2: Accelerated Query Performance

Since IDAA and Data Gate share large portions of the underlying code base, comparable performance results were expected with respect to the software stack. However, the cloud infrastructure abstraction layers (from OpenShift) and the new microservices architecture (introducing additional network communication between containers) may result in overhead reflected in the performance of Data Gate. The biggest performance impact is expected from the hardware resources and the Db2 service configuration that need to be specified when Data Gate is instantiated. While IDAA is preconfigured and tuned under lab conditions, Data Gate offers more options for its users. For the sake of space, just a single comparable configuration will be examined, without additional tuning on Data Gate side.

As testbed, a full-rack IDAA V7.5.8 on IBM Integrated Analytics System (IAS) cluster with 168 cores, 3.5 TB RAM, SSD storage, and 20 Gbps network connection to Db2/z was used. As counterpart, a Data Gate 2.1 on a IBM Cloud Pak for Data System (CP4DS) cluster with comparable hardware specification was used. Since the CP4DS offers more resources than the IAS cluster, the Db2 WH service was configured with less resources than the maximum to obtain a comparable target database. Data Gate uses Red Hat OpenShift Data Foundation (ODF) for storage which creates three replicas of each block that are distributed over multiple worker nodes. By using hostpath storage mapping, the performance could be increased because no data replicas are created and each worker node just mounts locally attached disks. However, since this configuration does not guarantee high availability, just the ODF results will be discussed. We used a 5 TB TPCB benchmark, extended by additional queries to match existing IDAA production workloads.

The performance results are outlined in Table 1. It can be seen that the initial load performance of Data Gate did not match expectations. The reason is a bottleneck in the network layer where additional tuning will be required. However, we do not consider this as a restriction

because tables are typically bulk-loaded just once and then incremental update is used. Hence, InSync performance is more important. Table 1 shows that both IDAA and Data Gate perform well in terms of throughput and latency during the incremental update process. We highlight that both IDAA and Data Gate were validated in real environments where the InSync pipeline could keep up with the maximum change rate of the corresponding Db2/z subsystem. The query results of the benchmark were also almost on par. The drill-down in the query timings in Figure 2 shows that most queries perform equally good in both environments. Some regressed while others performed better, both stemming from the configuration differences.

Overall, the assumptions were met. The experiments have shown that the architecture changes of Data Gate work well. In most cases the requirements of current Data Gate users are already satisfied. But the load performance may be improved with additional tuning.

## 6 Conclusion & Outlook

In this paper, we gave an introduction on Data Gate and demonstrated how it makes Db2/z data accessible to cloud applications by replicating and subsequently synchronizing the data with a cached twin in a Db2 on Cloud service. We discussed how Data Gate was built on IDAA technology, which parts of it could be reused, and which parts had to be adjusted and why. Our performance measurements revealed that classic IDAA use cases can be executed by Data Gate without major performance impact and identified bottlenecks that can be addressed by additional architecture tuning.

The evolution is not done, however. Today, the interfaces between Data Gate and IDAA are kept very similar which is not practical in the long run. Applications accessing the data in the cloud database should not have to use Db2/z to obtain information, like replication latency or details about synchronization errors. We are working on providing such administrative and monitoring information directly in the cloud database.

Another feature we are working on is to use the cloud database as a replication source. Organizations have expressed interest to selectively propagate data provided by Data Gate on to other database for further processing. Such a daisy-chain replication requires adjustments in how Data Gate stores the data in its cloud database because internal abstraction layers, like views that are used today, cannot serve as sources for replication tools like CDC [Be12].

A third major functional enhancement is to enable data modifications in the cloud database, propagating such changes back to the mainframe, and making other tables that exist in the cloud database known on the mainframe. Of course, such features will break the concept of treating Db2/z as master of the data. In this respect, security, consistency, and durability are important concerns that needs to be taken into account.

## Trademarks

IBM, DB2, and z/OS are trademarks of International Business Machines Corporation in the United States and/or other countries. Other company, product and service names may be trademarks, or service marks of IBM or other companies. All trademarks are copyright of their respective owners.

## Bibliography

- [Be12] Beaton, A.; Noor, A.; Parkes, J.; Shubin, B.; Ballard, C.; Ketchie, M.; Ketelaars, F.; Rangarao, D.; Tichelen, W.V.: . Smarter Business: Dynamic Information with IBM InfoSphere Data Replication CDC. IBM Redbooks, 2012.
- [BS17] Beier, Felix; Stolze, Knut: Architecture of a data analytics service in hybrid cloud environments. *it-Information Technology*, 59(3):151–158, 2017.
- [BSM16] Beier, Felix; Stolze, Knut; Martin, Daniel: Extending Database Accelerators for Data Transformations and Predictive Analytics. In: Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016. pp. 706–707, 2016.
- [Bu20] Butterstein, Dennis; Martin, Daniel; Stolze, Knut; Beier, Felix; Zhong, Jia; Wang, Lingyun: Replication at the speed of change: a fast, scalable replication solution for near real-time HTAP processing. *Proceedings of the VLDB Endowment*, 13(12):3245–3257, 2020.
- [Ch15] Chou, David C: Cloud computing: A value creation model. *Computer Standards & Interfaces*, 38:72–77, 2015.
- [Co22] Community, Debezium: . Debezium Documentation, 2022. <https://debezium.io/documentation/reference/stable/index.html>.
- [De21] Densmore, James: Data Pipelines Pocket Reference. O'Reilly Media, 2021.
- [DRD03] The Open Group. DRDA V5 Vol. 1: Distributed Relational Database Architecture, 2003.
- [Ge22] Ge, Zhiyu: Technologies and Strategies to Leverage Cloud Infrastructure for Data Integration. *Future And Fintech, The: Abcdi And Beyond*, p. 311, 2022.
- [Gu16] Gupta, Ravinder: Introduction to Oracle GoldenGate (OGG). In: *Mastering Oracle GoldenGate*. Apress, Berkeley, CA, pp. 3–10, 2016.
- [IB16] IBM: . Supported IBM Netezza Analytics stored procedures, 2016. <https://www.ibm.com/docs/en/daafz/5.1?topic=procedures-support-netezza-analytics-remote-stored>.
- [IB22a] IBM: . Archiving partition or table data with the High-Performance Storage Saver, 2022.
- [IB22b] IBM: . Db2 13 for z/OS documentation, 2022. <https://www.ibm.com/docs/en/db2-for-zos/13?topic=utilities-unload>.
- [IB22c] IBM: . IBM Cloud Pak for Data 4.5 documentation, 2022.

- [IB22d] IBM: . IBM DB2 Analytics Accelerator for z/OS 7.5, 2022.
- [IB22e] IBM: . IBM DB2 for z/OS Data Gate Analytics Accelerator for z/OS 7.5, 2022.
- [IB22f] IBM: . IBM Watson Knowledge Catalog 4.5.x documentation, 2022.  
<https://www.ibm.com/docs/en/cloud-paks/cp-data/4.5.x?topic=services-watson-knowledge-catalog>.
- [In20] Inc, TmaxSoft: Lift, shift and modernize: proven mainframe modernization strategies that enable digital transformation. 2020.
- [Je] Jeena, R; Saravanakumar, S; Bharathi, B Poornima; Priyanca, RP: Providing Consistency in Cloud Using Read after Write Technique to Endusers.
- [Kr11] Kreps, Jay; Narkhede, Neha; Rao, Jun et al.: Kafka: A distributed messaging system for log processing. In: Proceedings of the NetDB. volume 11, pp. 1–7, 2011.
- [MAT19] Muniswamaiah, Manoj; Agerwala, Tilak; Tappert, Charles: Data Virtualization for Decision Making in Big Data. *Int. J. Softw. Eng. Appl*, 10(5):45–53, 2019.
- [Me18] Mead, Larry: Microsoft:Migrating:Mainframe:Environments. 2018.
- [PP20] Paul, Ajo; Paul, Dipyaman: Migrating Mainframe workloads to Azure. Mindtree, 2020.  
<https://www.mindtree.com/sites/default/files/2020-11/Migrating-Mainframe-workloads-to-Azure-Whitepaper.pdf>.
- [SBM19] Stolze, Knut; Beier, Felix; Müller, Jens: Partial Reload of Incrementally Updated Tables in Analytic Database Accelerators. BTW 2019, 2019.
- [Sy19] Syniti: . Syniti Data Replication - User Guide, 2019.
- [Vo09] Vogels, Werner: Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

# The Easiest Way of Turning your Relational Database into a Blockchain — and the Cost of Doing So

Felix Schuhknecht<sup>1</sup> Simon Jörz<sup>2</sup>

**Abstract:** Blockchain systems essentially consist of two levels: The network level has the responsibility of distributing an ordered stream of transactions to all nodes of the network in exactly the same way, even in the presence of a certain amount of malicious parties (byzantine fault tolerance). On the node level, each node then receives this ordered stream of transactions and executes it within some sort of transaction processing system, typically to alter some kind of state. This clear separation into two levels as well as drastically different application requirements have led to the materialization of the network level in form of so-called blockchain frameworks. While providing all the “blockchain features”, these frameworks leave the node level backend flexible or even left to be implemented depending on the specific needs of the application. In the following paper, we present how to integrate a highly versatile transaction processing system, namely a relational DBMS, into such a blockchain framework to power a large variety of use-cases. As framework, we use the popular Tendermint Core, now part of the Ignite/Cosmos eco-system, which can run both public and permissioned networks and combine it with relational DBMSs as the backend. This results in a “relational blockchain”, which is able to run deterministic SQL on a fully replicated relational database. Apart from presenting the integration and its pitfalls, we will carefully evaluate the performance implications of such combinations, in particular, the throughput and latency overhead caused by the blockchain layer on top of the DBMS. As a result, we give recommendations on how to run such a systems combination efficiently in practice.

**Keywords:** Blockchain; Relational Databases; Distributed Query Processing; Tendermint

## 1 Introduction

In recent years, blockchain systems gained interest in various contexts, as they provide distributed transaction processing in potentially untrusted environments. Whereas the original applications mainly targeted public environments such as crypto currencies [Na09, Et22], blockchain systems have also gained interest in permissioned setups, where independent and potentially distrusting organizations, such as for instance companies trading with each other, want to perform some sort of mutual transaction processing [IB22a, IB22b, Te22a]. While the needs and environments for blockchain systems exist, a major downforce for the application of this technology has always been its hard entry level. Existing blockchain systems are often tailored towards a specific use-case or application domain and therefore are hard to apply for new application types. To deal with this challenge, one of the three following strategies is typically applied: (1) To reinvent the wheel and to engineer a new

---

<sup>1</sup> Johannes Gutenberg University Mainz, Institute of Computer Science, Staudingerweg 9, 55128 Mainz, Germany  
schuhknecht@uni-mainz.de

<sup>2</sup> sjoerz@students.uni-mainz.de

blockchain system from scratch, fitting to the specific needs. (2) To carefully adapt an existing blockchain system to the new requirements. (3) To not install a blockchain solution at all. Of course, often, consequence (3) is picked as (1) and (2) are cumbersome and therefore costly.

A step towards solving this problem is the observation that all blockchain systems essentially consist only of two major components. The first component manages the network level. It receives input transactions, orders them globally, and distributes the transaction sequence to each node of the network in exactly the same way. The challenge here lies in performing this in an untrusted environment, where a certain amount of participants might behave maliciously. To guarantee safety and liveness in such an environment, network levels implement sophisticated consensus mechanisms, secure message passing, and tamper-proof transaction logging. Despite various different implementations, the network level is rather independent from the actual application, as the semantics of the transactions are not relevant for this part. The second component manages the node level and centers around the processing of transactions within each node. Naturally, the requirements here are highly application dependent. As a consequence of these observations, blockchain *frameworks* have emerged that try to strictly separate their components by design. The prominent framework Tendermint Core [Te22b], that we will utilize in the following, even leaves the node level backend fully unimplemented. It is up to the application to provide a backend which receives and processes the transactions that are distributed by the framework to each node.

This allows us to easily tackle another typical downside of blockchain systems: an overly simplistic data model and low-level transaction logic. Many prominent systems, like the widely-used Hyperledger Fabric [An18], implement only a key-value model that is accessed via `put()/get()/delete()` calls, from a smart contract containing the transaction logic, often written in a general-purpose programming language like Go [An18]. Of course, this highly complicates the process of transaction writing. To tackle this problem, we want to support the widely-used relational model SQL, by connecting a relational DBMS as backend to the framework. Therefore, we create a “relational blockchain” with minimal effort and are especially interested in the overhead that is caused by this combination. We will investigate the latency and throughput of the relational blockchain under the drastically different synchronous, pseudo-synchronous, and asynchronous communication, each appropriate for different types of applications. Further, we will look at the scaling behavior of the system and discuss important configuration parameters. In summary, we will provide recommendations on how to use such a relational blockchain efficiently in practice.

## 1.1 Contributions

(1) We present how to integrate a stand-alone single-node relational DBMS into the blockchain framework Tendermint. Our current implementation supports PostgreSQL and MySQL and can easily be extended for further systems. As a result of this combination, we produce a *relational blockchain* that can execute (deterministic) SQL transactions equally across a set of potentially untrusted nodes to modify a fully replicated database.

- (2) We evaluate *latency* and *throughput/end-to-end runtime* of the relational blockchain under Smallbank [Sm13] and TPC-C [TP22] transactions. We compare its performance with a standalone execution of the workloads in single-instance and distributed PostgreSQL to identify the overhead that is caused by the blockchain framework.
- (3) We evaluate the impact of three different *communication methods*, namely synchronous, pseudo-synchronous, and asynchronous communication. We show that the choice of the communication method has a drastic impact on the performance of the system.
- (4) We evaluate the impact of the *relational backends*, namely PostgreSQL and MySQL, under synchronous and asynchronous communication.
- (5) We evaluate the *scaling capabilities* of the relational blockchain. Here, we first scale the number of virtual nodes within a physical node, which factors out network latency and resembles the Blockchain-as-a-Service (BaaS) setup. Then, we scale number of physical nodes within and across data-centers, resembling the classical distributed setup, facing network/internet latency.
- (6) We provide *practical recommendations* in which situations a relational blockchain yields a good performance – and in which situations it does not. To allow and easy application of our findings, all code, results, scripts and auxiliary material of this paper is available in the repository: <https://gitlab.rlp.net/fschuhkn/relational-blockchain>

## 2 Related Work

Before presenting our relational blockchain, let us discuss other work that sits at the intersection of blockchains and database systems.

There exists other interesting work that analyzes and/or builds upon the Tendermint framework. In [Ca21, Bu22], the authors perform an interesting performance analysis of the internal behavior of the framework. In [Am18], the authors analyze correctness and fairness of the system. The findings in these works justify our use and setup of Tendermint: The framework powers hundreds of applications of the Cosmos network, where most networks are tightly coupled with only few nodes. Latency and throughput decreases gracefully with the number of nodes participating in the consensus. Tendermint has also been used before to connect DBMSs as the backend. A prominent example is BigchainDB [Bi22], which uses the document store MongoDB [Mo22] as backend. Apart from Tendermint, there exist other blockchain frameworks. The most prominent representative is clearly Hyperledger Fabric [An18], designed to power permissioned blockchain networks. The modular design is composed of interchangeable components that allow a tuning of the network to the specific needs of the application up to a certain degree. Unfortunately, the system is hardcoded against a key-value model, such that the integration of a relational backend is not possible without deep changes of the system. Another blockchain framework is ChainifyDB [Sc21b], that allows the creation of heterogeneous blockchain networks. Here, heterogeneous means that different relational systems can be used across a single network. The applied processing model still ensures correctness of transaction processing.

Apart from frameworks, many research papers discuss the interconnection and relation of classical DBMSs and blockchain systems and how to combine both worlds. In BlockchainDB [E119b, E119a], a database layer is placed on top of a blockchain layer to combine the proper query interface of a database systems with the replication guarantees of a blockchain. In [Na19], the authors take the other route and extend a relational system, namely PostgreSQL, with a blockchain layer in order to create a blockchain network between multiple PostgreSQL instances. Unfortunately, this project requires a deep modification of PostgreSQL. Another interesting project is Veritas [Ge19]. Therein, the authors propose to extend existing DBMSs with blockchain features in a cloud environment. Apart from architectural works, many projects try to improve the performance of blockchain systems in order to converge towards the performance of traditional (distributed) DBMSs. In Fabric++ [Sh19], several optimization techniques from the database domain are transferred to Fabric in order to speed up processing. Other works try to improve blockchain performance via sharding [Da19] and various low-level optimizations in the transaction processing flow [Go19].

Note that originally, we planned to add a comparison with another comparable blockchain system to this paper to put our system into perspective. Unfortunately, there are very few systems targeting our specific setup and if they target it, they either (a) deeply modify the relational DBMS, (b) their code is not available, (c) have a very different query interface, or (d) run a different execution model providing different guarantees. Consequently, in this paper we focus on an in-depth evaluation of our relational blockchain system.

### 3 Setting up a Relational Blockchain

In the following section, we will discuss how to integrate a relational DBMS into the Tendermint framework, which we believe is a good template for how blockchain frameworks are reasonably engineered. On the backend side, we will focus on relational DBMSs in this work. However, the general process is applicable to non-relational transaction processing backends in a similar fashion.

#### 3.1 The Blockchain Framework: Tendermint Core

The design goal of the blockchain framework Tendermint Core [Te22b] is to provide essentially all those components that are shared in typical blockchain environments [Di18, Sc21a], but nothing more than that. Precisely, the entire transaction processing backend is left unimplemented and must be provided by the application side. There are two requirements for the backend: (1) The same backend must be used within all nodes of the network. (2) This backend must be deterministic, i.e., it executes a block of transactions in the same way on all nodes.

The most essential components that are already provided by Tendermint Core are:  
(1) A *transaction pool* which has the responsibility to receive and hold transactions that are pending for ordering and execution. All submitted input transactions first go into this

pool, where they can be rejected already, if they do not match user-specified criteria, by implementing the function `CheckTx()`. The pool itself is lazily replicated across the nodes, i.e., nodes share pending transactions with other nodes via gossip broadcasting.

(2) A *consensus mechanism* called Polka, which is a variation of the well-known PBFT [CL99] consensus. It can tolerate up to  $f$  maliciously behaving parties in a set of  $3f + 1$  parties. While the mechanism is tailored towards a permissioned setup, where all participants are known at all times, it can be extended to work in a public environment as well by using a Proof-of-Stake-like approach. As this requires the integration of a currency, we run the default version of the consensus mechanism in a permissioned setup.

(3) The *ledger*, which stores the observed sequence of committed transactions at the granularity of blocks within each node in a tamper-resistant way.

(4) A *message passing system* that ensures a secure communication between individual parties of the network.

On the network level, the workflow of the system essentially looks as follows: First, a client submits a new transaction to the network. The network then stores this transaction in the transaction pool with other pending transactions. A node then picks a set of transactions from the pool and groups them into a block in an ordered way. The block then goes through multiple consensus rounds until it is either globally accepted or globally rejected. If it is rejected, another block will be proposed (potentially by another node) and consensus restarts. However, if the block is accepted, it is distributed to all nodes of the network. Each node that receives a block then appends it to its copy of the ledger and passes the block to the transaction processing backend.

Apart from transaction processing, Tendermint Core also handles the network coordination such as the integration of new nodes to an already established network. A joining node essentially downloads the ledger from another node, verifies its integrity, and executes all blocks and their transactions in the backend to reach the up-to-date state.

### 3.2 Communicating with the Transaction Processing Backend

The block passing between the framework and the transaction processing backend happens via a so-called *Application Blockchain Interface (ABCI)*. The interface essentially consists only of the four functions `BeginBlock()`, `DeliverTx()`, `EndBlock()`, and `Commit()`, which must be implemented by the backend and which are called by the framework. For every agreed-upon block that is distributed, the core first calls `BeginBlock()` on each node to signal the arrival of a new block to the backend. Then, for each transaction within the block, the core calls `DeliverTx()` sequentially. This function is responsible for the actual processing of the transaction. It also returns whether the execution of a transaction was successful or not. After all transactions have been delivered, the core calls `EndBlock()` to signal that the block is done. Finally, the core calls `Commit()`. This tells the backend that all changes made by the transactions of the block must become real and visible for upcoming processing, if all transactions in the block succeeded. Otherwise, `Commit()` is responsible for rolling back all changes made by all transactions of the block. To implement

this ABCI and to connect a backend to Tendermint core, there are two options which we call the *server-variant* and the *builtin-variant*. In the server-variant, the backend implementing the ABCI runs as an independent socket-server and the core calls the interface via TCP. In the builtin-variant, the ABCI is implemented by the backend as a component of Tendermint core and directly compiled into it. While the server-variant offers a higher flexibility, the builtin-variant allows the core to communicate with the backend via simple function calls. In Section 4.2, we will evaluate both variants.

### 3.3 Integrating a Relational DBMS as Backend

Connecting a relational DBMS to the blockchain framework by implementing the ABCI is fairly natural, as both sides provide transaction semantics. However, to avoid confusion, we now have to clearly differentiate between different types of transactions and different types of commits in our system composition: We will call transactions, that are submitted to the blockchain network as *bc-transactions*. As discussed, multiple bc-transactions can be grouped in a block, which is committed as a whole by the framework. We call this a *bc-commit*. In contrast to that, we refer to transactions that are executed by the relational DBMS as *db-transactions*. The DBMS commits at the granularity of individual db-transactions, which we call *db-commit*.

Before being able to communicate with the relational DBMS from within the ABCI functions, we establish a connection to it in the bootstrapping part of Tendermint Core. To do so, we utilize the drivers `pgx` [pg22] and `go-sql-driver/mysql` [go22a] for PostgreSQL and MySQL, respectively, to open a connection to the DBMS instance. Table 1 now shows the pseudo-code implementation of `BeginBlock()`, `DeliverTx()`, and `Commit()`, where we show only the communication with the DBMS and removed any boilerplate code or error handling. As `EndBlock()` does not involve any DBMS communication, we do not show it.

```

1 BeginBlock() {
2   // start db-transaction
3   db-transaction dbTx
4   = db.Begin()
5   return dbTx
6 }
1 DeliverTx(db-transaction dbTx,
2           bc-transaction bsTx) {
3   // extract SQL statement
4   // from bc-transaction
5   sql stmt = DecodeTx(bsTx)
6   // execute SQL statement
7   // as part of db-transaction
8   status s = dbTx.Execute(stmt)
9   return s
10 }
1 Commit(db-transaction dbTx,
2        status[] s) {
3   if(s.Contains(bsTxFailed))
4     dbTx.Rollback()
5   else
6     // perform db-commit
7     // (= perform bc-commit)
8     dbTx.Commit()
9 }

```

Tab. 1: Pseudo-code for `BeginBlock()`, `DeliverTx()`, and `Commit()`.

In our implementation, `BeginBlock()` has the sole purpose to begin a new db-transaction. The context `db` is provided by Tendermint and implements a generic interface from the Go package `sql` [Go22b] that allows the communication with relational DBMSs. Relying on a generic interface enables an easy switching between PostgreSQL and MySQL (and other relational systems). Underneath this generic interface, we again use `pgx` respectively

`go-sql-driver/mysql` as a compatibility layer. It essentially translates the generic calls to their DBMS-specific counterparts. In each call to `DeliverTx()`, we receive the db-transaction in progress as well as a bc-transaction of the current block. We first decode the received bc-transaction and extract the SQL statement that is stored therein as a string. Then, we pass the SQL statement to the db-transaction context for execution. This execution returns a status (success or failure), which also contains the result of the db-transaction. We return this status to Tendermint Core. After all bc-transactions have been delivered, `Commit()` is called, which receives the open db-transaction and the execution statuses of all bc-transactions of the block. Based on the statuses, we check whether there is a bc-transaction that failed the execution. This could for instance be the case if the SQL statement contained in a bc-transaction is malformed. If a failed bc-transaction exists, we command the DBMS to rollback the db-transaction, including all changes made by bc-transactions of the block. Otherwise, we can safely db-commit the db-transaction, such that all changes of this block become visible for the processing of the next block.

Note that we use the previously described communication protocol only for *modifying* transactions. To answer read-only transactions, we implement the ABCI function `Query()` which allows us to fire read-only queries against the backend of a single node<sup>3</sup>, effectively bypassing the costly transaction processing flow of the blockchain framework.

### 3.4 Synchronous vs Asynchronous Transaction Processing

To process modifying transactions in the blockchain network, the client has essentially two different modes available: (1) A *synchronous mode*, where a client-request blocks until it receives an answer from the system. (2) An *asynchronous mode*, where the request returns before receiving an answer. As we will evaluate both modes in the following, let us discuss their precise realization and behavior in the following.

We start with synchronous processing. First of all, to communicate with Tendermint Core, the client uses a Broadcast API in order to submit bc-transactions. From this API, we utilize the function `BroadcastTxCommit()`. This function basically resembles a synchronous submit that receives a bc-transaction and blocks until either it has been worked into a bc-committed block or it is rejected from the transaction pool (due to being malformed). Consequently, our test suite looks fairly simple for the synchronous case: In each iteration of the loop, a client fires a bc-transaction using `BroadcastTxCommit()` and waits for the result before proceeding with the next iteration. The asynchronous transaction processing is more complex. Here, we use the weaker API function `BroadcastTxSync()` to communicate with Tendermint Core, which already returns after the bc-transaction has been successfully added to the transaction pool. Thus, the client does not get a synchronous response on whether the transaction was committed successfully in a block or not. As we still require a reliable feedback on the success of execution, we implement a test suite as depicted in Figure 1.

---

<sup>3</sup> This can be extended to query multiple nodes to handle the risk of querying a malicious node.

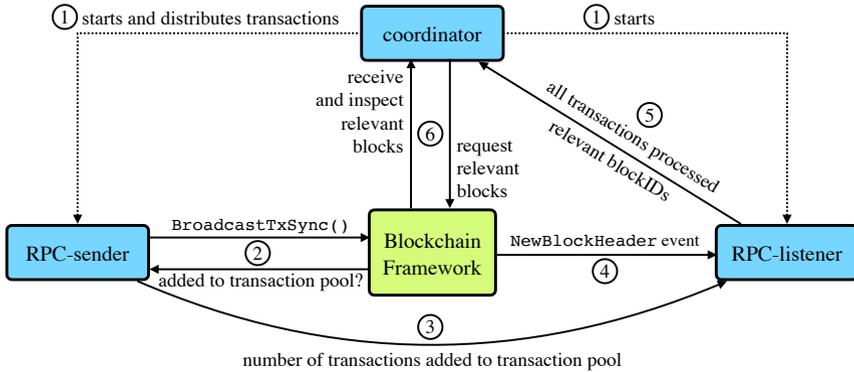


Fig. 1: Workflow of asynchronous transactions processing.

It consists of three components: (a) The *coordinator*, responsible for orchestrating the entire run. (b) The *RPC-sender*, which broadcasts the bc-transactions to the framework. (c) The *RPC-listener*, which listens for bc-committed blocks. In ①, the main loop first starts both RPC-sender and RPC-listener. Then, in ②, the RPC-sender uses the aforementioned `BroadcastTxSync()` to push bc-transactions into the network. While doing so, the RPC-sender monitors the number of bc-transactions that made it into the transaction pool – this is the number of transactions expected to make it through the system. In ③, after submitting all bc-transactions, this number is passed to the RPC-listener. For every block that is bc-committed by the framework, in ④, the RPC-listener receives a `NewBlockHeader` event from the framework and calculates the number of already seen bc-transactions based on it. As soon as it has seen all previously entered bc-transactions, in ⑤, it informs the coordinator that all bc-transactions have now been processed and passes the blockIDs containing these transactions. In ⑥, the coordinator then requests all relevant blocks and checks whether the bc-transactions have been processed successfully. Note that the steps ② and ④ can happen interleaved, i.e., the sender can still push in new bc-transactions while the listener is already receiving headers of committed blocks.

### 3.5 Deterministic Execution, Error Handling, and Provided Guarantees

As the blockchain framework essentially resembles a fully replicated state machine, it requires the backend to behave deterministically, which we ensure in two steps:

(1) The repetitive calls to the ABCI function `DeliverTx()` by the framework happen sequentially. As we ensure that any communication with the relational DBMS within `DeliverTx()` happens synchronously, all bc-transactions will be executed within a db-transaction in exactly the same order within the relational DBMS of each node. Note that there exist mechanisms [Sc21b] to process transactions concurrently *and* deterministically in the backend of all nodes that could be used here. However, as we will see in the experimental evaluation, the backend-execution is dominated by Tendermint Core, and therefore, such an optimization would not lead to significant performance improvements. Consequently, we

kept the execution sequential. (2) We submit only bc-transactions that contain deterministic SQL statements, similar as done in the related work [E119b, E119a]. This requires stripping SQL from components such as random-number generators, timestamp functions, and the LIMIT-statement.

Orthogonal to that, our framework supports the (optional) computation and comparison of checksums on the state after each block commit to detect any occurred state deviation. However, this check comes with a significant overhead and is currently not practical in performance-critical production systems. Also note that a state deviation would require to be followed by a roll-back of the block that caused the deviation in order to recover. Such a recovery is currently not supported by our system, similar to the situation for other blockchain systems. Instead, a deviating node is excluded from the network. In terms of transactional safety, any communication with the relational DBMS happens through db-transactions. This also holds for read-only queries. Therefore, read-only queries are also guaranteed to see a consistent state (resembling the state after a block commit).

## 4 Experimental Setup

Before starting with our actual experimental evaluation and analysis, let us discuss the setup in the following. As blockchain systems are used in different setups, we will evaluate different network configurations. First, to completely factor out network latency overhead, we perform a set of experiments on a single powerful machine equipped with an Intel i9-12900K CPU (Alder Lake) running at up to 5.2GHz with 16 cores. This state-of-the-art processor is able to run a set of virtual nodes and simulates a very low latency blockchain network. The machine contains 128GB of DDR4-3200. All database files are located on a 2TB Samsung 980 Pro M2 PCIe 4.0 SSD. As operating system, a 64-bit Arch Linux is installed. Note that such a setup consisting of a single physical node running the blockchain network is not fully artificial nowadays: So called Blockchain-as-a-Service (BaaS) solutions [So22, AEE21] host all virtual nodes of the network in a single data-center, often also on the same physical node. Second, to measure the impact of a distributed setup across the internet, we also perform an additional set of experiments on a network of up to eight AWS EC2 instances (t2.small), which are distributed across the four regions Frankfurt, Ireland, London, and Paris, with up to two instances per region. Each instance has one vCPU, contains 2GB of RAM, has 16GB of gp2 volume attached (general purpose SSD), and runs Ubuntu 20.04. Additionally, in the Frankfurt data-center, we run a separate instance (t2.micro) that serves as the client and orchestrates our runs. Note that for all experiments, each client establishes a single connection that is kept alive and re-used during the benchmark run to keep the communication overhead as low as possible.

### 4.1 Benchmarks: Smallbank & TPC-C

In the following evaluation, we use transactions and datasets from two established transactional benchmarks from the world of blockchains and databases, namely Smallbank [Sm13] and TPC-C [TP22]. We use transactions from these two benchmarks as they offer very

different characteristics: While Smallbank contains a set of five extremely simple and short-running transactions which essentially resemble only money transfers between accounts, the three used TPC-C transactions are far more complex and long-running. In the following, we give a brief overview of the used benchmarks.

For Smallbank, the database consists of a single table with four columns, where each tuple contains a user-ID and a name having both a balance for a checking account and a savings account, initialized with random integers. We use the five modifying transactions that are specified in the original benchmark description. The transactions `TransactSavings` and `DepositChecking` each increase the respective account balance. `SendPayment` modifies two checking account balances. `WriteCheck` decreases a checking account balance. Finally, `Amalgamate` moves money from a savings account to the checking account of the same user. For each transaction, we randomly pick the account(s) as well as the amount to modify/move following a uniform distribution. For TPC-C, the database has nine tables in total and essentially represents a multi-warehouse wholesale operation. We implement the two modifying transactions `NewOrder` and `Payment` and select the parameters of each fired transaction randomly within meaningful bounds as specified by the TPC-C benchmark description. Additionally, we implement the read-only transaction `OrderStatus` to test the query-interface of the framework. We selected these three transactions as they are rather complex by accessing all nine tables and by modifying five of them, resulting in more long-running transactions than for Smallbank. The warehouses and districts are accessed by the transactions following a uniform distribution. Note that all Smallbank transactions are transmitted to the system on-the-fly. In contrast to that, the TPC-C transactions are registered in the relational DBMS as stored procedures due to their significantly higher code complexity and size. The transactions of TPC-C then simply contain a call of the corresponding stored procedure.

## 4.2 Framework and Backend Configuration

For Tendermint Core, we use the latest stable version 0.34 for all experiments. For PostgreSQL, we use version 14.5, for MySQL, we run version 8.0.30. Tendermint Core, PostgreSQL, and MySQL run in Docker containers and are installed from the corresponding Docker images. Tendermint Core as well as the relational DBMS are deeply configurable. To measure the “out-of-the-box” performance, we start with the default configuration and try to tune the systems as little as possible. We state and justify in the following all changes.

On the side of Tendermint Core, we first increase the size of the transaction pool from 5,000 to 100,000 transactions, such that the whole transaction sequence of each benchmark always fits in. Next, there are multiple timeout parameters that have an impact on both the performance and the behavior of the network. We set `timeout_broadcast_tx_commit` to a sufficiently large value (10s), such that synchronous communication never times out in our experiments. Also, we have to tune the important parameter `timeout_commit`, which determines how long the consensus mechanism does wait for additional votes, if 2/3 of the votes have been received already. To empirically identify a good value, in Figure 2, we

perform an experiment where we vary `timeout_commit` from 25ms to 1000ms for both 1,000 synchronous and 10,000 asynchronous transactions of Smallbank. As a value of 100ms yields the best end-to-end runtime in both cases, we use a timeout of 100ms in all upcoming experiments. Further, we set the maximum allowed block size to 21MB such that the size is never the limiting factor for forming a block. We disable the creation of empty blocks (in case the transaction pool runs dry) as well. For PostgreSQL and MySQL, we essentially keep the configuration of the used Docker images as is.

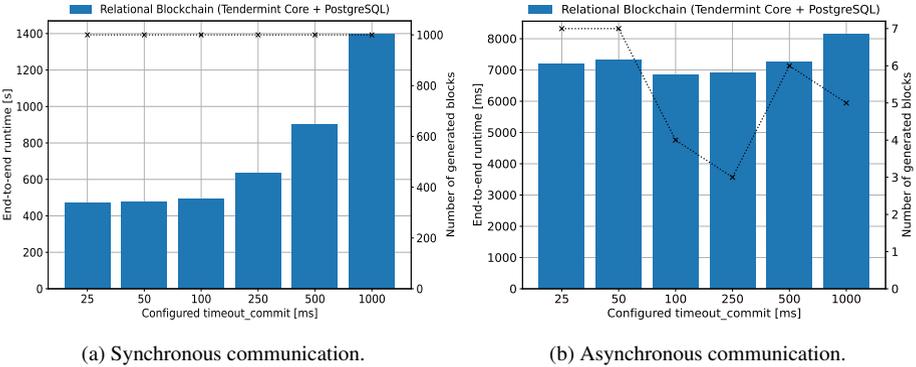


Fig. 2: Varying the `timeout_commit` parameter from 25ms to 1000ms.

As mentioned in Section 3.2, there are two ways of connecting the backend to the framework, where the server-variant is more flexible than the builtin-variant. To identify the performance impact, we implemented both variants and evaluate them against Smallbank and TPC-C transactions. Figure 3 shows the results for both synchronous and asynchronous communication. We can see that for synchronous communication, there is hardly a difference

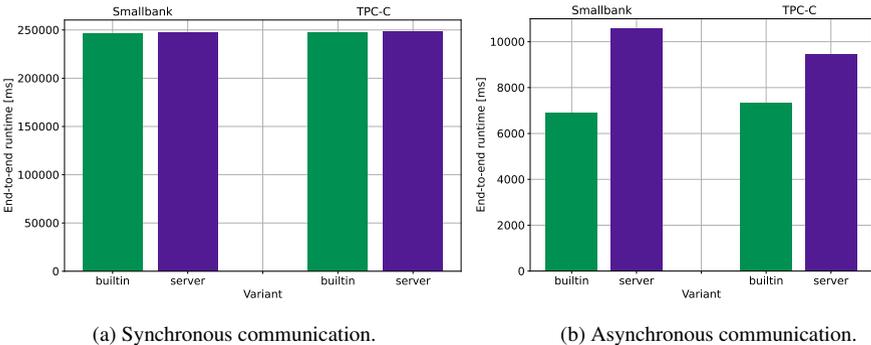


Fig. 3: Server-variant vs builtin-variant.

visible. The type of connecting the backend is fully overshadowed by the high cost of synchronous communication (which we will evaluate in detail in Section 5.1.1). However, for the cheaper asynchronous communication, the builtin-variant is 1.53x faster for Smallbank

and 1.23x faster for TPC-C than the server-variant. This is due to the fact that in the server-variant the ABCI calls happen via TPC sockets, which are significantly more expensive than the direct function calls in the builtin-variant. Due to the higher performance, we use the builtin-variant in all following experiments.

## 5 Experimental Evaluation & Analysis

In the following, we perform a set of experiments to determine the performance of the relational blockchain. We are particularly interested in its overhead (Section 5.1) over the raw relational DBMS. Then, we perform a cost breakdown to see where the time actually goes (Section 5.2). Next, analyze the impact of the number of clients (Section 5.3) and the relational DBMS (Section 5.4). Then, we investigate the scaling capabilities of the network (Section 5.5). Finally, we analyze the overhead of our relational blockchain over a distributed PostgreSQL cluster (Section 5.6).

### 5.1 Overhead of the Blockchain Framework

We start our experimental evaluation with the central question of how much overhead the blockchain frameworks actually adds on top of the relational DBMS. We compare the performance of our relational blockchain setup (Tendermint Core + PostgreSQL) as described previously, with the performance of the raw (single-instance) DBMS (PostgreSQL only). Note that the idea of this is not to compare our relational blockchain with PostgreSQL, but to measure the overhead of the blockchain framework over the backend.

As setup we use a fairly typical permissioned configuration: We have a single client firing the bc-transactions into a network of four virtual nodes, where each node consists of a Tendermint Core instance as well as a PostgreSQL instance, each running in its docker container. Again, we use virtual nodes here to factor out any network latency. We test two workloads: (1) The previously described TPC-C workload with 10 warehouses. (2) The Smallbank workload with 100,000 accounts. Note that to distinguish the transactions of the workload from bc-transactions/db-transaction, we will call the former *wl-transactions* in the following. As the type of communication impacts cost and usability, we perform in the following a synchronous as well as an asynchronous variant of the experiment.

#### 5.1.1 Synchronous and Pseudo-synchronous Communication

We start with synchronous communication. It basically resembles the typical communication with a DBMS: A client submits a transaction to the system and the call blocks until eventually, it returns the result. As we have described in Section 3.4, the blockchain framework supports such a communication style via its Broadcast-API. Additional to this fully synchronous communication, where one *wl-transaction* is packed into one *bc-transaction*, we also test a pseudo-synchronous communication style. Therein, we pack multiple *wl-transactions* into a single *bc-transaction* and fire this *bc-transaction* synchronously. On one hand, this results in fewer *bc-transactions* that have to go through the system, potentially lowering the pressure

on the network and the transaction processing overhead. On the other hand, this relaxes our notion of synchronicity (hence pseudo), as the client receives a synchronous response only for a batch of wl-transactions, not for each wl-transaction individually.

To asses the overhead, we are interested in both the *latency* and the *end-to-end runtime*. In this context, latency is the time between submitting a bc-transaction and receiving a response to it. Note that we submit a new bc-transaction only after receiving a response to the previous one. The end-to-end runtime is the time between submitting the first bc-transaction and receiving the response to the last bc-transaction. Figure 4 and Figure 5 show the results for Smallbank and TPC-C, respectively, where we fire a uniform mixture of 1,000 writing wl-transactions in total. On the *x*-axis, we vary the number of wl-transactions per fired bc-transaction from 1 to 2,048 in logarithmic steps. As discussed, 1 resembles the synchronous case, whereas 2 to 2,048 resemble different pseudo-synchronous configurations. On the *y*-axis, we show in the Figures 4a and 5a the average latency of a wl-transaction over the whole transaction sequence. In Figures 4b and 5b, we show the end-to-end runtime on the *y*-axis. To improve readability, we use a logarithmic scale on the *y*-axis.

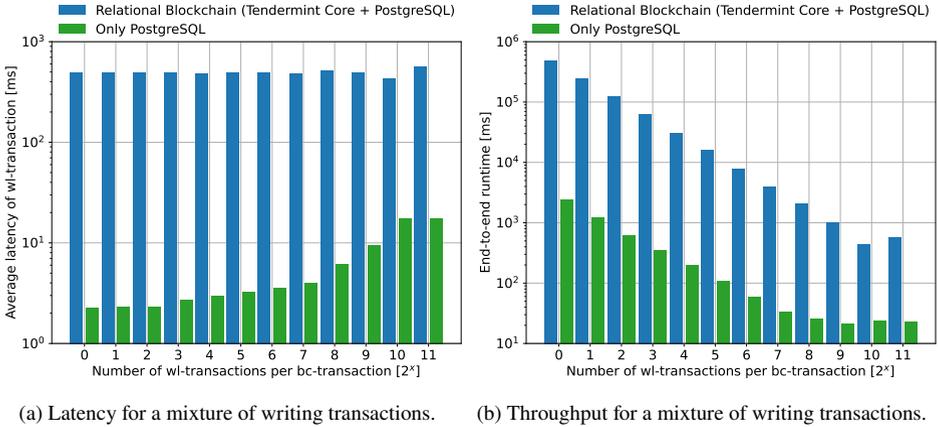
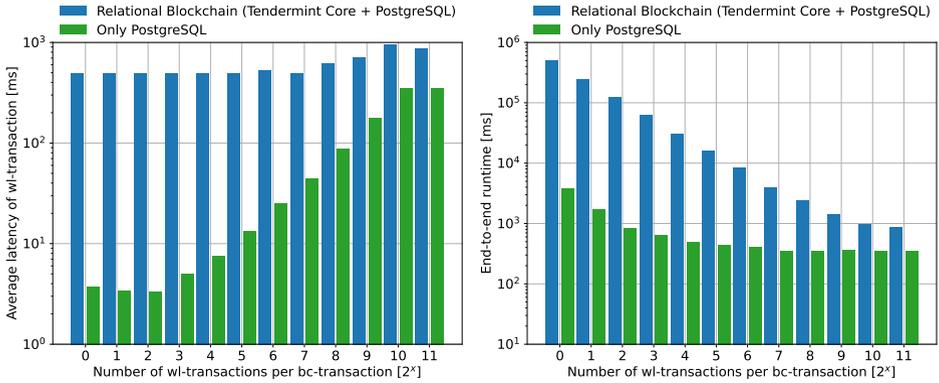


Fig. 4: Synchronous communication (Smallbank)

In the results, we can observe a significant overhead of the blockchain framework over the relational DBMS under both workloads and all synchronicity configurations. However, we can also see that the overhead depends on (a) the type of workload and (b) the number of wl-transactions packed into a single bc-transaction, i.e., the amount of required synchronicity.

Regarding (a), we can see that under Smallbank (Figure 4), the overhead of the blockchain framework over the raw relational DBMS is much more significant than under TPC-C (Figure 5). While for Smallbank, the smallest observed overhead is a still a slowdown of 32x and 25x in latency and end-to-end runtime, respectively, for TPC-C, the latency and runtime overhead decreases to only 2.5x and 2.4x in the best case. The reason for this lies in the complexity and individual runtime of the wl-transactions. As complex TPC-C transactions



(a) Latency for a mixture of writing transactions. (b) Throughput for a mixture of writing transactions.

Fig. 5: Synchronous communication (TPC-C)

require more processing time in the relational backend than the short-running Smallbank transactions, the overhead of the framework makes a smaller fraction of the total runtime.

Regarding (b), we observe that packing multiple wl-transactions into a single bc-transaction heavily impacts the performance, both for the relational blockchain and the raw relational DBMS. While for the fully synchronous case, we measure a devastating overhead of 218x (latency) and 208x (end-to-end runtime) for Smallbank and 133x (latency) and 129x (end-to-end runtime) for TPC-C, the situation gradually improves when relaxing the required synchronicity. Particularly for TPC-C, a reduction in synchronicity has a positive impact on the amount of overhead introduced by the framework, which decreases to the aforementioned acceptable 2.5x (latency) and 2.4x (end-to-end runtime). This is due to the fact that less blocks are formed for the transaction sequence, requiring less consensus rounds, decreasing the central bottleneck of the blockchain framework.

### 5.1.2 Asynchronous Communication

Let us now look at the asynchronous case, which resembles the typical communication style with a blockchain system: The client submits a bc-transaction and the submission returns immediately. Then, after some time, the client checks whether the transaction has been bc-committed or not (yet).

Here, ensuring a fair experimental setup between the relational blockchain and the standalone DBMS is a bit more complicated. The reason lies in the way Tendermint Core handles asynchronous transaction processing internally: As we do not have to wait for a response, we push the whole batch of wl-transactions into the network in one go. Tendermint Core then forms blocks out of pending wl-transactions and commits them one after the other. As previously described, for each block, an individual db-transaction is opened and eventually

committed, each containing a sequence of applied wl-transactions. However, as Tendermint now decides by itself how many wl-transactions it packs into a single block, it is more difficult to set up a comparable run for the standalone DBMS. To solve the problem, we record the transactions that were packed in each committed block during the run of the relational blockchain. Then, to set up the run with standalone PostgreSQL, we pack the exact same transaction sequences in individual db-transactions and fire them one by one.

Figure 6 and Figure 7 show the experimental results. As asynchronous communication is less expensive than synchronous communication in total, we fire a larger sequence of 10,000 wl-transactions this time. In the Figures 6a and 7a, we show the measured *latency* of

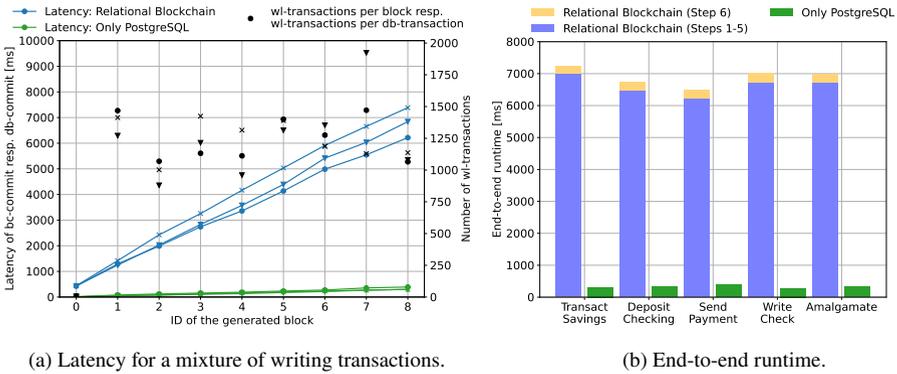


Fig. 6: Asynchronous communication (Smallbank)

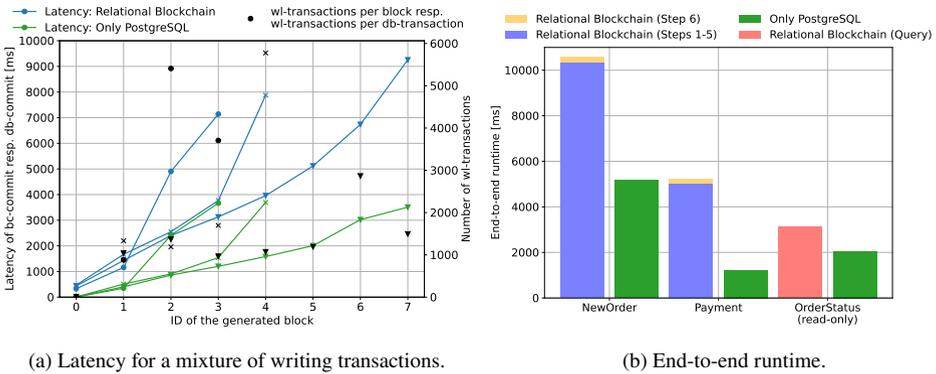


Fig. 7: Asynchronous communication (TPC-C)

each block respectively fired db-transaction. Here, we measure latency as the time between the start of the experiment (firing the first wl-transaction) until the notification about the bc-commit of the respective block (step 4 in Figure 1). We fire a uniformly selected mix of only writing transactions of the respective benchmark and plot the ID of each block that has been generated on the *x*-axis in relation to the latency of the corresponding bc-commit

on the y-axis. Additionally, we plot the number of wl-transactions that were packed by the framework in each individual block (blacks dots) with respect to a second y-axis. As for individual runs, the framework might produce a different number of blocks, we plot each of the three performed runs individually.

Additionally, in the Figures 6b and 7b, we perform a set of experiments where we measure the *end-to-end runtime*. In this case, we fire 10,000 transactions of each type individually to analyze an effect of the transaction type. For the relational blockchain, we split the end-to-end runtime for the sequence of modifying transactions into the actual transaction processing time (steps 1 to 5 of Figure 1) and the time to check whether the transaction has been processed successfully (step 6 of Figure 1). For the read-only transaction `OrderStatus` of TPC-C, we show the runtime when using the query-interface of the framework.

Let us first look at the results for Smallbank in Figure 6. We can see that the difference in latency and end-to-end runtime between the relational blockchain and stand-alone PostgreSQL is significant. Processing the transactions in the framework increases the latency of the last generated block respectively db-commit by up to 24x and the end-to-end runtime by an average of 21x over all transactions. We see that the result inspection (step 6) is not responsible for the overhead, the actual transaction processing in the framework takes the majority of time. We can also see that the framework packs around 1,000 to 2,000 wl-transactions in one block, leading to the generation of 8 blocks in total. This clearly improves the performance over the synchronous case, however, still generates significant overhead. Between the individual transaction, we observe little difference. All transactions are extremely short-running in the backend and modify at most two accounts each.

Let us now inspect the TPC-C results in Figure 7, which look quite different to the results of Smallbank. First of all, we can see that the overhead of the framework over stand-alone PostgreSQL is significantly smaller for this benchmark. This time, the framework increases the latency at most by 2.6x. The end-to-end runtime of the sequence of `NewOrder` and `Payment` transactions increases only by 2.0x and 4.3x, respectively. The reason lies in the much higher complexity of the performed transaction: If the backend requires more time to process a transaction, the overhead of processing it in the framework becomes less significant in the end-to-end runtime. For the read-only transaction, the overhead of the framework is even smaller with 1.52x, as we can bypass block forming and consensus entirely. This shows that for queries, the framework should be bypassed entirely. Overall, we can also see that the overhead under asynchronous communication is significantly smaller than in the synchronous case.

## 5.2 Cost Breakdown

To get a deeper insight on where the overhead originates from, we analyze the produced logfiles of Tendermint Core and isolate four individual phases: (1) The proposal phase, in which pending transactions are grouped in a block to propose. (2) The consensus phase, in which consensus on the proposed block is performed. (3) The execution phase, in which

the bc-transactions of the block are executed against the backend. (4) The commit, that marks the state change. Figure 8 shows the life a block under synchronous and asynchronous

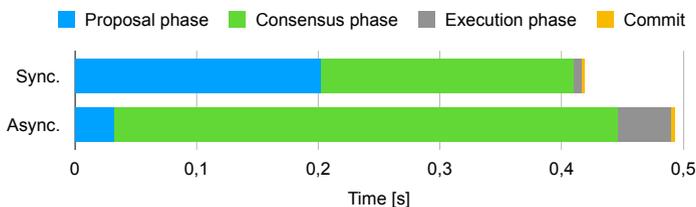
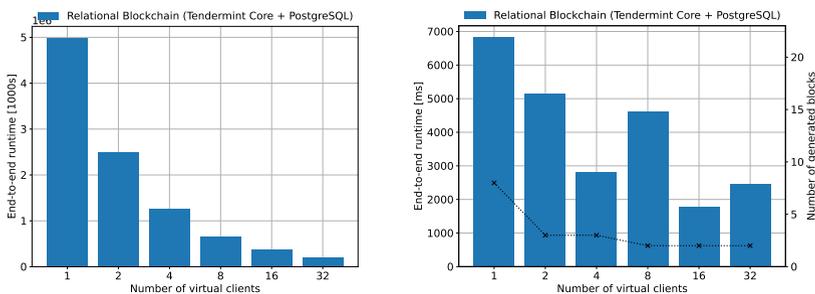


Fig. 8: Cost breakdown into individual phases.

communication. In the synchronous case, it contains one bc-transaction, whereas in the asynchronous case, 1311 bc-transactions are packed in the block. We can see clearly that the consensus phase is by far the dominating phase of the pipeline. In both cases, the actual execution is negligible in comparison. Also, we can observe that the runtime of the proposal and consensus phase varies across runs.

### 5.3 Impact of the Number of Clients

Let us now inspect the impact of the number of clients on the end-to-end runtime of the system. In Figure 9, we vary the number of clients firing transactions from 1 to 32 in logarithmic steps while keeping the total number of Smallbank transactions fixed to 10,000.



(a) Synchronous communication.

(b) Asynchronous communication.

Fig. 9: Impact of the number of clients.

We can see that in the synchronous case, the performance drastically increases with the number of clients. This is the case as concurrently submitted transactions are now packed in the same block. For asynchronous communication, the performance improvement is naturally smaller, but also significant, showing that a single client does not saturate the system.

## 5.4 Impact of the Relational Backend

So far, we used PostgreSQL as the relational DBMS in the backend for all experiments. Let us now investigate whether the choice of the relational system actually matters or whether its performance is completely overshadowed, if it is part of the blockchain framework. In Figure 10, we show the end-to-end runtime of our relational blockchain under a uniform mixture of 1,000 synchronous respectively 10,000 asynchronous wl-transactions of Smallbank, where we use either PostgreSQL or MySQL as the backend in all four nodes.

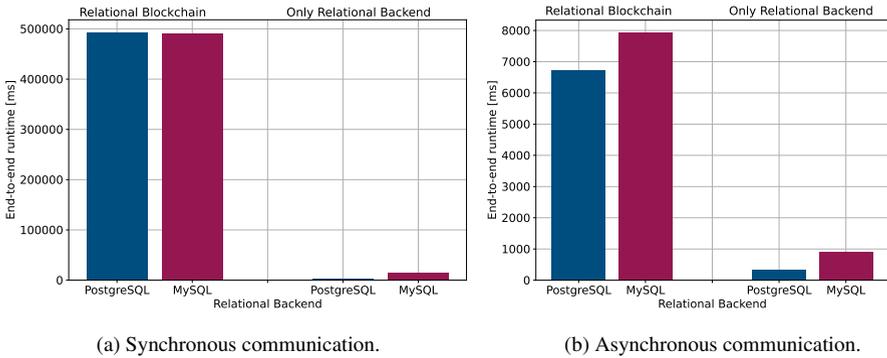


Fig. 10: Impact of the relational DBMS in the backend.

Let us first look at the raw backend performance shown on the right side of the plots. We can see that PostgreSQL is able to process the sequence of transactions significantly faster than MySQL. In the synchronous case, PostgreSQL is 6.6x faster than MySQL. In the asynchronous case, where multiple wl-transactions are packed in a single db-transaction, the speedup is still 2.6x. While the backends perform drastically different, this difference becomes less significant when embedding the backend within the relational blockchain. In the synchronous case, the backend makes no difference at all, as the runtime is dominated by block forming and consensus. Only in the asynchronous case, we see a significant difference. Therein, using PostgreSQL improves the end-to-end runtime by 1.2x over MySQL.

## 5.5 Impact of Scaling across Virtual Nodes and Physical Nodes

Until now, we ran all experiments using four virtual nodes running on one physical node. In the following, we will vary both the number of virtual nodes (Figure 11a) as well as the number of physical nodes (Figure 11b) to represent the network.

We start by varying the number of virtual nodes in Figure 11a. This experiment still factors out network latency. We set up a network of only one virtual node, four virtual nodes, and eight virtual nodes and report the end-to-end runtime for 10,000 modifying Smallbank transactions using asynchronous communication. Additionally, we show the number of generated blocks for the total run. Note that a network consisting of only one node can skip the consensus phase, as no other participants exist to coordinate with. We see this

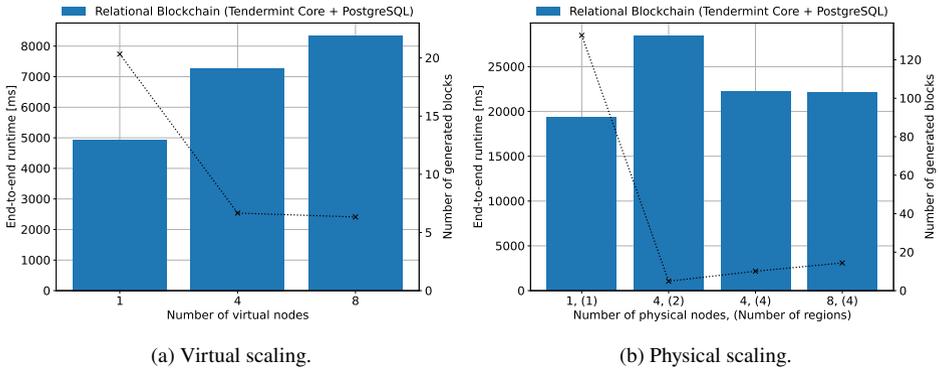


Fig. 11: Scaling the number of virtual and physical nodes.

setup as the baseline for the throughput that can be achieved in the system. When looking at the results in Figure 11a, we can see that the end-to-end runtime is unsurprisingly the shortest when running only one node. When using four nodes, the runtime increases by a factor of 1.47x over the single node configuration, when using eight nodes, it increases by a factor of 1.69x. This shows that an increase in the number of nodes clearly increases the overhead, however, only sublinearly. When inspecting the number of generated blocks, we can see that for one node, 20 (smaller) blocks are generated on average, whereas for four and for eight nodes, only 6 (larger) blocks are generated for the whole sequence of 10,000 transactions. This shows that the consensus that is performed for a block throttles the forming of the next block.

Let us now look at the results when scaling the number of physical nodes in Figure 11b. Here, we test one physical node in one region (Frankfurt), four physical nodes in two different regions (Frankfurt and Paris), four physical nodes in all four different regions, and eight physical nodes in all four different regions. Note that for this experiment, we repeat each run 10 times (instead of 3 times as before) to factor out variance caused by the cloud provider as much as possible. First of all, we can observe that the end-to-end runtime is overall higher than when scaling the number of virtual nodes within one physical node. This is caused by the internet latency, but also by the slower physical nodes. Again, using only one node is unsurprisingly fastest, however, using more physical nodes does not decrease the performance as heavily as for virtual scaling. Using four physical nodes within two regions shows worse performance than four physical nodes within four regions. We deduce from this that the internet traffic between the nodes is not the bottleneck here, but that the two physical nodes within the same region potentially share the same hardware resources. Going to eight physical nodes decreases the performance only marginally in comparison to four nodes. We also observed a relatively high variance between individual runs as soon execute on a distributed setup. For four physical nodes on two regions, we measured runtimes between 22s and 37s, for four physical nodes across four regions, we observed runtimes between 20s and 25s, and for eight physical nodes, we saw runtimes between 21s and 28s. This indicates that other computations happened on the same instances.

## 5.6 Comparison with a Distributed Relational DBMS

Let us finally investigate the overhead of our system in comparison with a fully-replicated distributed PostgreSQL cluster using Citus [Cu21] across our four EC2 nodes. We fire our typical set of Small-bank transactions against the coordinator node. This coordinator uses 2PC to synchronize all modifications with the three replicas. In comparison, we install our relational blockchain on the same nodes. Figure 12 shows the results for a varied pseudo-synchronous communication. We can see that for few wl-transactions per bc-transaction, distributed PostgreSQL performs drastically better. However, the fewer bc-transactions are formed, the more the performance of our relational blockchain approaches distributed PostgreSQL.

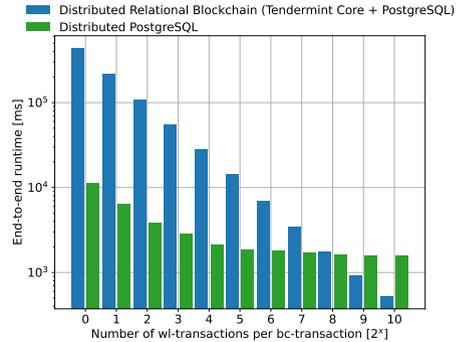


Fig. 12: Comparison of our relational blockchain with a fully-replicated distributed PostgreSQL.

## 6 Takeways and Conclusion

In this work, we have presented a practical and feasible way of integrating a full-fledged relational DBMSs into a blockchain framework to support the execution of deterministic SQL transactions in byzantine environments. We analyzed the performance implications of such a systems combination and identified situations where the overhead is acceptable. Also, we have seen setups where the overhead is dramatic and completely overshadows the backend performance. In Table 2, we conclude with practical recommendations on how to achieve the best performance of such a setup.

| Property      | Recommendation  |
|---------------|---|
| Communication | If possible, chose asynchronous communication. Alternatively, chose pseudo-synchronous communication with as many wl-transactions per bc-transaction as acceptable.   |
| Clients       | Use several clients to propose bc-transactions (improvement till up to 32 clients), especially under synchronous communication.   |
| Backend       | If the workload contains complex transactions and communication is asynchronous, chose a high-performance backend (e.g. PostgreSQL over MySQL). Otherwise, the choice of backend is less important.             |
| Transactions  | Fire transactions as read-only transactions if possible to bypass the transaction processing flow of the framework.   |
| Network       | Use as few nodes as possible to keep the overhead of the consensus phase and the communication between the nodes as low as possible. Across physical nodes, the system scales better than across virtual nodes. |

Tab. 2: Practical recommendations on how to achieve good performance.

## Bibliography

- [AEE21] Alshurafa, Samer Muneer; Eleyan, Derar; Eleyan, Amna: A survey paper on blockchain as a service platforms. *Int. J. High Perform. Comput. Netw.*, 17(1):8–18, 2021.
- [Am18] Amoussou-Guenou, Yackolley; Pozzo, Antonella Del; Potop-Butucaru, Maria; Tucci Piergiovanni, Sara: Correctness and Fairness of Tendermint-core Blockchains. *CoRR*, abs/1805.08429, 2018.
- [An18] Androulaki, Elli; Barger, Artem; Bortnikov, Vita; Cachin, Christian; Christidis, Konstantinos; Caro, Angelo De; Enyeart, David; Ferris, Christopher; Laventman, Gennady; Manevich, Yacov; Muralidharan, Srinivasan; Murthy, Chet; Nguyen, Binh; Sethi, Manish; Singh, Gari; Smith, Keith; Sorniotti, Alessandro; Stathakopoulou, Chrysoula; Vukolic, Marko; Cocco, Sharon Weed; Yellick, Jason: Hyperledger fabric: a distributed operating system for permissioned blockchains. In (Oliveira, Rui; Felber, Pascal; Hu, Y. Charlie, eds): *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*. *ACM*, pp. 30:1–30:15, 2018.
- [Bi22] BigchainDB: <https://www.bigchaindb.com/>, September 2022.
- [Bu22] Buchman, Ethan; Guerraoui, Rachid; Komatovic, Jovan; Milosevic, Zarko; Serebinschi, Dragos-Adrian; Widder, Josef: Revisiting Tendermint: Design Tradeoffs, Accountability, and Practical Use. In: *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Supplemental Volume, Baltimore, MD, USA, June 27-30, 2022*. *IEEE*, pp. 11–14, 2022.
- [Ca21] Cason, Daniel; Fynn, Enrique; Milosevic, Nenad; Milosevic, Zarko; Buchman, Ethan; Pedone, Fernando: The design, architecture and performance of the Tendermint Blockchain Network. In: *40th International Symposium on Reliable Distributed Systems, SRDS 2021, Chicago, IL, USA, September 20-23, 2021*. *IEEE*, pp. 23–33, 2021.
- [CL99] Castro, Miguel; Liskov, Barbara: Practical Byzantine Fault Tolerance. In (Seltzer, Margo I.; Leach, Paul J., eds): *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*. *USENIX Association*, pp. 173–186, 1999.
- [Cu21] Cubukcu, Umur; Erdogan, Ozgun; Pathak, Sumedh; Sannakkayala, Sudhakar; Slot, Marco; Citus: Distributed PostgreSQL for Data-Intensive Applications. In (Li, Guoliang; Li, Zhanhuai; Idreos, Stratos; Srivastava, Divesh, eds): *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. *ACM*, pp. 2490–2502, 2021.
- [Da19] Dang, Hung; Dinh, Tien Tuan Anh; Loghin, Dumitrel; Chang, Ee-Chien; Lin, Qian; Ooi, Beng Chin: Towards Scaling Blockchain Systems via Sharding. In (Boncz, Peter A.; Manegold, Stefan; Ailamaki, Anastasia; Deshpande, Amol; Kraska, Tim, eds): *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. *ACM*, pp. 123–140, 2019.
- [Di18] Dinh, Tien Tuan Anh; Liu, Rui; Zhang, Meihui; Chen, Gang; Ooi, Beng Chin; Wang, Ji: Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Trans. Knowl. Data Eng.*, 30(7):1366–1385, 2018.
- [El19a] El-Hindi, Muhammad; Binnig, Carsten; Arasu, Arvind; Kossmann, Donald; Ramamurthy, Ravi: BlockchainDB - A Shared Database on Blockchains. *Proc. VLDB Endow.*, 12(11):1597–1609, 2019.

- [El19b] El-Hindi, Muhammad; Heyden, Martin; Binnig, Carsten; Ramamurthy, Ravi; Arasu, Arvind; Kossmann, Donald: BlockchainDB - Towards a Shared Database on Blockchains. In (Boncz, Peter A.; Manegold, Stefan; Ailamaki, Anastasia; Deshpande, Amol; Kraska, Tim, eds): Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019. ACM, pp. 1905–1908, 2019.
- [Et22] Ethereum Yellow Paper, <https://gavwood.com/paper.pdf>, September 2022.
- [Ge19] Gehrke, Johannes; Allen, Lindsay; Antonopoulos, Panagiotis; Arasu, Arvind; Hammer, Joachim; Hunter, James; Kaushik, Raghav; Kossmann, Donald; Ramamurthy, Ravi; Setty, Srinath T. V.; Szymaszek, Jakub; van Renen, Alexander; Lee, Jonathan; Venkatesan, Ramarathnam: Veritas: Shared Verifiable Databases and Tables in the Cloud. In: 9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings. [www.cidrdb.org](http://www.cidrdb.org), 2019.
- [Go19] Gorenflo, Christian; Lee, Stephen; Golab, Lukasz; Keshav, Srinivasan: FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. In: IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019. IEEE, pp. 455–463, 2019.
- [go22a] go-sql-driver/mysql - MySQL Driver, <https://github.com/Go-SQL-Driver/MySQL/>, October 2022.
- [Go22b] Golang sql package, <https://pkg.go.dev/database/sql>, September 2022.
- [IB22a] IBM Food Trust: <https://www.ibm.com/de-de/blockchain/solutions/food-trust>, September 2022.
- [IB22b] IBM Health: <https://www.ibm.com/de-de/blockchain/industries/healthcare>, September 2022.
- [Mo22] MongoDB: <https://www.mongodb.com/>, September 2022.
- [Na09] Nakamoto, Satoshi: Bitcoin: A Peer-to-Peer Electronic Cash System. May 2009.
- [Na19] Nathan, Senthil; Govindarajan, Chander; Saraf, Adarsh; Sethi, Manish; Jayachandran, Praveen: Blockchain Meets Database: Design and Implementation of a Blockchain Relational Database. Proc. VLDB Endow., 12(11):1539–1552, 2019.
- [pg22] pgx - PostgreSQL Driver and Toolkit, <https://github.com/jackc/pgx>, September 2022.
- [Sc21a] Schuhknecht, Felix Martin: Talking Blockchains: The Perspective of a Database Researcher. In: 37th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2021, Chania, Greece, April 19-22, 2021. IEEE, pp. 72–75, 2021.
- [Sc21b] Schuhknecht, Felix Martin; Sharma, Ankur; Dittrich, Jens; Agrawal, Divya: chainifyDB: How to get rid of your Blockchain and use your DBMS instead. In: 11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings. [www.cidrdb.org](http://www.cidrdb.org), 2021.
- [Sh19] Sharma, Ankur; Schuhknecht, Felix Martin; Agrawal, Divya; Dittrich, Jens: Blurring the Lines between Blockchains and Database Systems: the Case of Hyperledger Fabric. In (Boncz, Peter A.; Manegold, Stefan; Ailamaki, Anastasia; Deshpande, Amol; Kraska, Tim,

eds): Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019. ACM, pp. 105–122, 2019.

- [Sm13] Smallbank Benchmark, <https://hstore.cs.brown.edu/documentation/deployment/benchmarks/smallbank/>, May 2013.
- [So22] Song, Jie; Zhang, Pengyi; Alkubati, Mohammed; Bao, Yubin; Yu, Ge: Research advances on blockchain-as-a-service: architectures, applications and challenges. *Digit. Commun. Networks*, 8(4):466–475, 2022.
- [Te22a] Telekom Blockchain Applications: <https://dmexco.com/stories/how-deutsche-telekom-uses-blockchain/>, September 2022.
- [Te22b] Tendermint Core, <https://tendermint.com/core/>, September 2022.
- [TP22] TPC-C Benchmark, <https://www.tpc.org/tpcc/>, September 2022.



## Session 2



# WannaDB: Ad-hoc SQL Queries over Text Collections

Just tell it what you want, what you really, really want

Benjamin Hättasch,<sup>1,2,3</sup> Jan-Micha Bodensohn,<sup>1,2</sup> Liane Vogel,<sup>1,2</sup> Matthias Urban<sup>2</sup> and Carsten Binnig<sup>2,3</sup>



**Abstract:** In this paper, we propose a new system called *WannaDB* that allows users to interactively perform structured explorations of text collections in an ad-hoc manner. Extracting structured data from text is a classical problem where a plenitude of approaches and even industry-scale systems already exist. However, these approaches lack in the ability to support the ad-hoc exploration of texts using structured queries. The main idea of *WannaDB* is to include user interaction to support ad-hoc SQL queries over text collections using a new two-phased approach. First, a superset of information nuggets from the texts is extracted using existing extractors such as named entity recognizers. Then, the extractions are interactively matched to a structured table definition as requested by the user based on embeddings. In our evaluation, we show that *WannaDB* is thus able to extract structured data from a broad range of (real-world) text collections in high quality without the need to design extraction pipelines upfront.

**Keywords:** interactive text exploration; text to table; matching embeddings

## 1 Introduction

A question like “What were the days with a COVID-19 incidence rate higher than 750 in Germany?” can be answered with a simple SQL query if the relevant information is present in a database. Yet, in case there are only written (i.e., textual) reports available such as those published by governmental organizations like the RKI in Germany,<sup>4</sup> the situation is much more complex: answering such queries over collections of textual documents that each contain only a part of the information needed requires that first the relevant attributes are extracted from each document, before they are stored in a structured form (i.e., a spreadsheet or a database table) in order to make them available for structured queries.

One could now argue that extracting structured data from text is a classical problem for which there is a plethora of approaches and where even several industry-scale systems already exist: for example, DeepDive [Sa16] that was acquired by Apple or System-T

<sup>1</sup> Primary Authors

<sup>2</sup> Technical University of Darmstadt, Systems Group, 64287 Darmstadt, Germany, firstname.lastname@cs.tu-darmstadt.de

<sup>3</sup> DFKI

<sup>4</sup>[https://www.rki.de/DE/Content/InfAZ/N/Neuartiges\\_Coronavirus/Situationsberichte/Gesamt.html](https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Gesamt.html)

```
SELECT report_date WHERE incidence_rate > 500;  
SELECT region, AVG(incidence_rate) GROUP BY region HAVING AVG(incidence_rate) > 500;  
SELECT AVG(vaccinated_twice) WHERE report_date > 21-01-01 AND report_date < 21-02-01;
```

Fig. 1: Exemplary ad-hoc information needs phrased as SQL-like queries in *WannaDB*. Two classes of ad-hoc queries are supported: Queries that extract facts from individual documents (e.g., first query) as well as queries that involve aggregation and grouping (e.g., the latter two queries).

[Le20a] from IBM are such systems that have developed rather versatile tool suites to extract structured facts from textual sources. However, these systems require a team of highly-skilled engineers that compile extraction pipelines, which often includes training particular machine learning models, and then populate a structured database from the given text collection. And even more importantly, the resulting extraction pipelines are typically static and can only be used to extract a pre-defined (i.e., fixed) set of attributes and tables for a certain text collection. This prevents exploratory scenarios where users can ask ad-hoc queries regardless of whether a pipeline has been set up to extract the attribute or not.

Hence, being able to ad-hoc execute SQL-like queries over a text collection without the need to manually compose extraction pipelines would be a major step forward compared to existing approaches for structured data extraction from text. Use cases with needs for such ad-hoc structured querying of unstructured text can be found in various domains beyond the example mentioned before, e.g., data scientists together with medical doctors looking for new insights through medical reports or data journalists examining hundreds of documents as part of their investigations. Structured queries provide a higher expressiveness (e.g., aggregation and filtering operations), and more rigorousness in the calculation of the results compared to the usage of natural language queries in classical question answering systems.

**Contributions.** In this paper, we hence propose *WannaDB*, a system that can execute SQL-like queries on text collections in an ad-hoc manner. Examples for queries that *WannaDB* supports can be found in Figure 1. Overall, *WannaDB* supports two classes of queries: (1) *Ad-hoc Fact Queries*: queries that extract facts from text documents to construct table rows. This also involves applying filter predicates and projection operations, as shown by the first query in Figure 1. (2) *Ad-hoc Aggregate Queries*: queries that in addition involve aggregations and grouping over multiple documents as shown by the two other queries in Figure 1, which come with additional challenges like named entity disambiguation/cross-document co-reference resolution that we discuss later in this paper. *WannaDB* can therefore directly produce tables stating information that is not explicitly mentioned in the documents and hence not discoverable by pure extraction or search approaches. To enable such ad-hoc SQL queries over a given text collection, *WannaDB* implements a novel extraction and querying pipeline that builds on two key ideas:

The first key idea of *WannaDB* is that, different from existing approaches which aim to extract information for a specific (i.e., fixed) information need from a given text collection, *WannaDB* instead implements a *holistic extraction* approach that aims to extract a wide spectrum of information from a given text collection (called information nuggets in the

sequel). For this holistic extraction, *WannaDB* implements a framework approach and relies on a set of different general-purpose extraction methods, such as approaches for named-entity recognition. Moreover, during extraction, *WannaDB* computes embeddings for all the information nuggets, taking several signals such as the textual mentions itself, and the position in the text into account.

As a second key idea, to answer ad-hoc queries on top of the extracted information nuggets, *WannaDB* implements a novel *interactive matching* approach that aims to map the information nuggets to the information needs specified by the user in form of an SQL query: embeddings of the extracted information nuggets together with the embeddings of the query attributes are used to decide which information nuggets qualify for answering the query. For this matching, *WannaDB* requests feedback from the user whether certain information nuggets are the correct values for the required query attributes. The system carefully selects these requests to minimize the amount of required feedback. The query attributes can be of a much finer granularity than the labels of the extraction approaches used in the first stage (e.g., `airline` instead of `ORG`) and *WannaDB* can even distinguish between similar attributes with just a small semantic difference (e.g., the amounts of people vaccinated once and twice).

While other approaches that can extract tables from text such as learned sequence-to-sequence models [WZL22] often suffer from a phenomenon called hallucination (i.e., they generate values that are not in the actual source document), our approach can guarantee that the contents of the produced result tables always originate from the queried documents. Moreover, compared to learned question answering approaches, *WannaDB* can perform numerical reasoning on the data without the need to rely on the limited mathematical abilities [He21] of a language model.

In order to evaluate the abilities of *WannaDB*, we conduct a wide range of experiments on text collections from different domains ranging from aviation reports over daily COVID-19 situation reports to multiple text collections created from Wikipedia that cover different categories (Nobel laureates, countries, and skyscrapers). We show that *WannaDB* not only outperforms other baselines that can be used for ad-hoc query answering on text collections, but is also competitive with approaches that are trained or refined on domain-specific data. Moreover, our evaluation shows that typically only a few interactions per query attribute are sufficient to answer a query over hundreds or thousands of source documents. Overall, answering an SQL query over text documents with *WannaDB* (by providing minimal interactive feedback) only takes a few minutes, compared to hours and hours of manually extracting information or refining an extraction pipeline without *WannaDB*. Finally, to make the results reproducible, we will make our source code and the data sets used for evaluation available at <https://link.tuda.systems/wannadb>.

**Outline.** Next, we describe the functions of *WannaDB* in an exemplary usage scenario, before we explain the different components in Section 3. In Section 4, the algorithms behind the interactive components are discussed in further depth, followed by a short overview of

the current limitations in Section 5. We provide an evaluation of *WannaDB* in Section 6 and an overview of existing and related work in Section 7, before we conclude in Section 8.

## 2 Exemplary Usage

In this exemplary usage scenario, we aim to show how *WannaDB* can be used to satisfy an information need based on a text collection. Imagine, e.g., a data journalist who just obtained a large collection of airline incident reports and is now looking for noticeable events, like a high rate of incidents for a certain carrier or airport. They use *WannaDB* for that purpose. The data journalist starts by loading the collection of text files into *WannaDB* for processing and triggers the pre-processing of the files, a process that needs to be done only a single time for each text collection.

Next, the data journalist enters an SQL-like query as a starting point for their exploration (e.g., `SELECT airline, airport, COUNT(*) GROUP BY airline, airport`). As there is no pre-existing table yet, the `FROM`-part of a typical SQL query can be omitted, simplifying the query syntax. After entering the query, *WannaDB* presents a list of possible matches for each required attribute (e.g., airline) found in texts of the collection, as shown in Figure 2. Not all the found matches will be correct right away, therefore *WannaDB* relies on some user input to adjust the results. The data journalist confirms a few of the correctly found matches, corrects wrong matches by choosing the relevant extraction or marks if the required attribute does not occur in a given text (see Figure 2). Meanwhile, *WannaDB* continuously updates the list of all guessed matches during this interactive phase, leveraging the feedback. The user interface allows to quickly identify entries that stand out and get an impression of the quality already achieved. Once the data journalist is satisfied with the quality of the matches, they continue with the next attribute of their query.

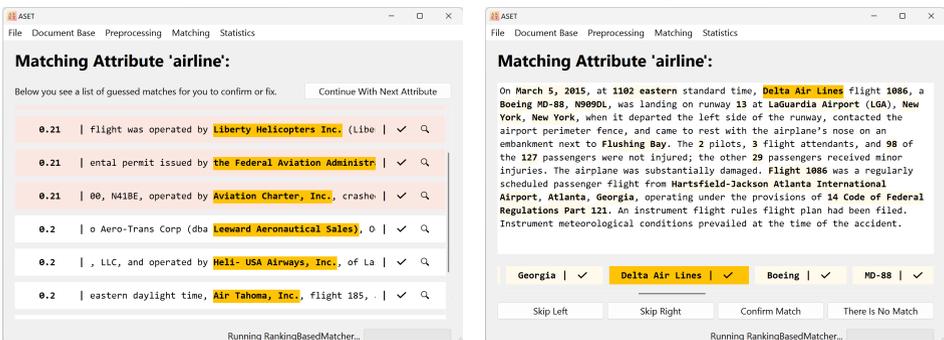


Fig. 2: Graphical user interface of *WannaDB*, more details can be found in our SIGMOD’22 demo [HBB22]. Left: potential matches over and under the threshold are shown, the user is asked to either confirm or fix them. Right: Inspect a document and fix by selecting the correct match.

After all attributes are processed, *WannaDB* will execute the query on the resulting table. If the query contains grouping operations, the data journalist might be asked again for some interactive feedback (e.g., to confirm that *Lufthansa* and *LH* refer to the same airline, but *LHS* does not). *WannaDB* will again try to transfer this feedback to other rows. In the end, the data journalist will receive an answer to their query and can export the resulting table to a spreadsheet, an SQLite table, or a Pandas Dataframe for further investigation. If they have further queries to submit to *WannaDB*, the interactive matching process only needs to be repeated for new attributes, as *WannaDB* leverages existing results from previous queries.

### 3 System Overview & Architecture

In this section, we describe the architecture of *WannaDB*. It consists of two stages: an offline stage to extract information nuggets (i.e. short information-bearing text snippets), followed by the interactive stage to answer the query by table extraction and if required interactive filtering or grouping. The overall workflow is visualized in Figure 3. Here, we give an overview of both stages and the relevant components of *WannaDB*. More details of the table extraction as well as grouping and filtering, which are the main contributions of *WannaDB*, are described in Section 4.

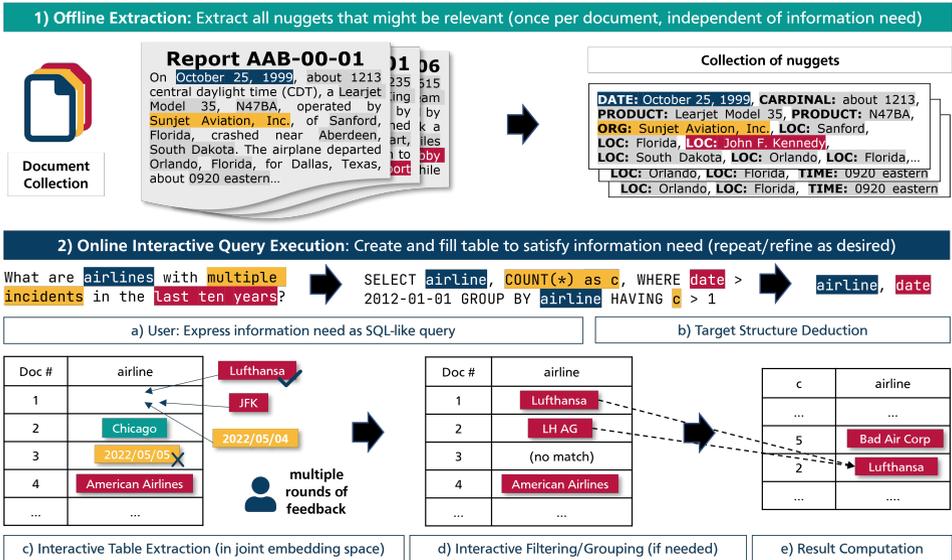


Fig. 3: Architecture & exemplary usage: The offline extraction phase obtains information nuggets from the documents. The online phase then infers the required structure from a query, matches between the extracted information nuggets and the user’s schema, performs the grouping and executes the query.

### 3.1 Stage 1: Offline Extraction

In the first stage we employ off-the-shelf information extractors to extract a superset of potentially relevant information nuggets (e.g., named entities) from the given text collection. This step is independent of user queries and can thus be executed offline to prepare the text collection for ad-hoc exploration by the user. The extractors process the collection document-by-document to generate the corresponding extractions. Clearly, a limiting factor of *WannaDB* is which kinds of information nuggets can be extracted in the extraction stage, since only this information can be used for the subsequent matching stage. As a default, we use named entity recognizers from *Stanza* [Qi20] and *spaCy* [Ho20]. In general, *WannaDB* can be used with any extractor that produces label-mention pairs; i.e. a textual mention of an information nugget in the text (e.g., *American Airlines*) together with a natural language descriptor representing its semantic type called label (e.g., *Company*). Moreover, additional information about the extraction (e.g., its position in the document and the surrounding sentence) is also stored and used for computing the embeddings, as we describe below.

After extraction, the information nuggets are pre-processed to derive their actual data values (i.e., a canonical representation, e.g., for timestamps) from their mentions. For this we also rely on state-of-the-art systems for normalization [Ma14]. The nuggets are then represented based on the following signals: (1) *label* – the entity type determined by the information extractor (e.g. *Company*),<sup>5</sup> (2) *mention* – the textual representation of the entity in the text (e.g., *Lufthansa*), (3) *context* – the sentence in which the mention appears, (4) *position* – the position of the mention in the document. Each information nugget representation comprises embeddings for the individual signals (1-4). We compute semantic representations for the natural language signals using FastText [Mi18] (1), Sentence-BERT [RG19] (2) and BERT [De19] (3) and normalize the position by dividing it by the document length.

### 3.2 Stage 2: Interactive Query Execution

At runtime, a user issues queries and interacts with the system. *WannaDB* infers the table structure required to answer a query, and employs a novel interactive matching stage to map the information nuggets extracted in the first stage to the required query attributes.

**Interactive Table Extraction.** The first step of the interactive query execution of *WannaDB* is the interactive table extraction from the text documents. In this step, a table with attributes is filled by *WannaDB* to answer a given user query. The required table structure is automatically inferred from the user’s SQL query. *WannaDB* checks which attributes are mentioned explicitly as attributes to return, and as part of aggregation operations, or implicitly in filter predicates or group-by statements. Then, *WannaDB* starts to fill the table with the derived schema by executing the interactive table extraction algorithm.

---

<sup>5</sup>We map the named entity recognizers’ labels like *ORG* to suitable natural language expressions according to the descriptions in their specification.

In the interactive table extraction, the user interacts with *WannaDB* in order to fill the required attributes of the result table with the information nuggets extracted before. To find matching nuggets, *WannaDB* first computes embeddings for the target attributes similar to the ones computed for the information nuggets in the offline phase.

A classical approach to determine a mapping between information nuggets and attributes of the user table would be to train a machine learning model in a supervised fashion to classify to which attribute the extracted information nugget should be mapped to. However, learning such a classification model would require a substantial set of labeled training data for each attribute and thus prevent ad-hoc queries. Instead, our approach leverages embeddings to quantify the intuitive semantic closeness between information nuggets and the attributes of the user table. For the attributes of the target table, only the attribute names are available to derive an embedding, while for the extracted nuggets we can make use of more information as we described above.

*WannaDB* therefore employs a novel interactive matching strategy that incorporates user feedback and operates in the joint embedding space of nuggets and target attributes. This strategy works in an attribute-by-attribute fashion and collects user feedback (e.g., confirming or correcting a possible match). *WannaDB* uses distances between possible and confirmed matches to populate the remaining cells. This process is steered by carefully selecting potential matches that are presented to the user for feedback to reach a high matching quality with as little feedback as possible.

**Interactive Filtering & Grouping.** After the interactive table extraction step, *WannaDB* executes the interactive filtering and grouping stage for answering a user query. Remember, *WannaDB* has the aim to work on text collections from domains without pre-existing resources like refined language models or custom knowledge bases. Grouping and filtering the extracted table thus is challenging, since it is filled with mentions from the text directly, hence applying these operations might lead to faulty query results if entities are not correctly resolved: e.g., the table might contain entries such as *Deutsche Lufthansa* and *German Lufthansa Airline* which both refer to the same entity. Applying `GROUP BY` or a `WHERE` directly on such an extracted table would return multiple lines (i.e., one for each different mention even though they refer to the same entity). *WannaDB* therefore again uses interaction to perform those operations on the level of embeddings instead of string representations, as will be described in detail in the next section.

## 4 Interactive Query Execution

*WannaDB* introduces novel embedding-based algorithms for interactive table extraction as well as filtering and grouping. In this section, we describe these algorithms in further detail (see Figure 4 for a pseudo-code representation).

## 4.1 Interactive Table Extraction

In the interactive table extraction stage, *WannaDB* populates the attributes of the table one by one. To fill the cells of a certain attribute, *WannaDB* aims to select one matching information nugget from each of the documents. To do so, *WannaDB* associates each information nugget with a *cached distance* that corresponds to the certainty with which it believes that the nugget matches the attribute. For each document, *WannaDB* considers the information nugget with the lowest cached distance as the document's currently *guessed match*. Furthermore, *WannaDB* uses a distance *threshold* for each attribute to decide when a cell should be left empty instead. The details of how this threshold is calculated and interactively adapted are explained in Section 4.2. The overall procedure of the table extraction is shown in Figure 4.

In the beginning, each nugget's cached distance is initialized as the cosine distance between the nugget's label embedding (e.g., *Organization*) and the embedding of the attribute name (e.g., *Airline*) (Figure 4, line 2-3). After initialization, the interactive feedback phase starts. *WannaDB* presents a ranked list of documents with their currently guessed matches to the user for feedback (see Figure 2) and will continuously update the list after every given feedback. This allows the user to quickly identify (incorrect) entries that stand out and to get an impression of the quality already achieved. The ranked list is centered around the threshold and thus hopefully shows both correct guesses with a low certainty, and incorrect guesses, where *WannaDB* would profit most from feedback.

The user can then provide feedback for any of these guesses (line 7): they may either confirm the guess, select another information nugget from the document, or state that the document does not contain a matching information nugget. In case their feedback results in a confirmed match, this matching information nugget is used to update the cached distances of all other remaining information nuggets (line 13-16). To compute the distance between two information nuggets, *WannaDB* calculates the mean of the cosine distances between their individual signal embeddings. The distance updates ensure that a nugget's cached distance is always the distance to the closest confirmed match. Considering distances between information nuggets allows *WannaDB* to capitalize on more signals like the textual mentions (e.g., *American Airlines*) of other matching information nuggets.

Next, *WannaDB* updates the documents' currently guessed matches by selecting the information nuggets with the lowest cached distances (line 21). Finally, *WannaDB* then adjusts the threshold accordingly (see Section 4.2 for more details). Moreover, the user can at any time decide to terminate the interactive feedback phase and continue with the next attribute. All remaining documents' cells without explicitly confirmed matches will then be populated with their currently guessed matches (line 24-28) if there is at least one with a distance that is low enough (i.e., below the threshold).

```

1  for attribute in query.attributes: # Process each attribute separately
2  for nugget in all_nuggets:
3      nugget.distance = compute_distance(attribute, nugget) # Compute initial distances
4
5  while interactive_feedback_phase: # Interactively get user feedback
6      ranked_list = make_ranked_list(threshold, documents)
7      feedback = get_user_feedback(ranked_list)
8      match feedback:
9          # Positive feedback (confirmation or manually correction):
10         case ConfirmNugget(document, confirmed_nugget):
11             # Mark this particular cell as manual confirmed...
12             set_match(document, confirmed_nugget)
13             # ... and update distances for all nuggets based on user feedback
14             for nugget in all_nuggets:
15                 new_distance = compute_distance(nugget, confirmed_nugget)
16                 nugget.distance = min(new_distance, nugget.distance)
17         # Negative feedback:
18         case NoMatchInDocument(document):
19             # Direct effect only on the given document...
20             leave_empty(document)
21     update_guessed_matches(documents)
22     adjust_threshold(feedback) # ... but both feedback types can have effects indirectly
23     ↪ through threshold adjustment on other document's rows, too
24
25 for document in documents: # Only consider values up to a given maximum distance
26     if current_guess(document).distance < threshold:
27         set_match(document, current_guess(document)) # compute final result table
28     else:
29         leave_empty(document)
30
31 def adjust_threshold(feedback): # Feedback can be further exploited in certain cases
32     match feedback:
33         case ConfirmNugget(document, confirmed_nugget):
34             if confirmed_nugget.distance > threshold:
35                 increase_threshold(confirmed_nugget)
36         case NoMatchInDocument(document):
37             if current_guess(document).distance < threshold:
38                 decrease_threshold(document)
39
40 def decrease_threshold(document): # Consider fewer matches as valid (especially those
41     ↪ above last marking as incorrect that are currently accepted nevertheless)
42     nuggets = ranked_list.between(threshold, document)
43     min_dist = min(n.distance for n in nuggets)
44     threshold = min(min_dist, threshold)
45
46 def increase_threshold(confirmed_nugget): # Consider more matches as valid (especially
47     ↪ those below last confirmation that are currently discarded because of the threshold)
48     nuggets = ranked_list.between(confirmed_nugget, threshold)
49     max_dist = max(n.distance for n in nuggets)
50     threshold = max(max_dist, threshold)

```

Fig. 4: Pseudo-Code representation of our interactive algorithm for table extraction, including threshold adjustment.

## 4.2 Threshold Adjustment

*WannaDB* uses a threshold for two purposes: (a) to decide when it is better to leave a cell empty than to use a very unlikely guess (mostly because the desired value is not mentioned in the document) and (b) to select guesses to present to the user where feedback will have as much effect as possible. This threshold is automatically tuned during the runtime of *WannaDB* to fit the data at hand. Given the approximate query setting *WannaDB* is built for, we decided to use a common threshold for all regions forming in the embedding space instead of individually tuning it, to keep the number of interaction cycles low.

The adjustment of the threshold is shown in Figure 4 (line 30–47). The general idea is to incorporate the additional knowledge gained from the user confirming a nugget even though it was above the threshold or correcting an entry below the threshold. This feedback action will only affect a certain nugget directly, but other similarly well fitting nuggets from other documents might still be accepted or discarded wrongly because of the threshold, which is therefore carefully adapted after feedback actions: If the user confirms a nugget from the ranked list that is above the threshold, all nuggets between the threshold and this nugget should be considered as a good guess. In the case that any of the nuggets is still above the threshold after the calculation of the new distances, the threshold is adapted accordingly. In contrast, if the user states that for a nugget with a distance below the threshold there is no match in the document, the threshold is decreased to also exclude other matches that are in the list above the nugget if necessary. The threshold is only adapted in these two cases, where implicit hints about the quality assessment by the user can be incorporated.

## 4.3 Interactive Filtering & Grouping

In the following, we explain how interactive grouping is supported in *WannaDB* to tackle the problem of different surface forms for the same entries. Filtering works similarly, but we omit the details due to space limitations.

To resolve entities correctly, the interactive grouping algorithm is based on agglomerative clustering using the distances between the information nugget embeddings for an attribute. Entries with the same string representation are merged without interaction. For the remaining ones, the different signals from the extraction phase are utilized. *WannaDB* presents all distinct members of two clusters that should potentially be merged to the user and asks them to confirm whether these all describe the same entity. If that is the case, the clusters are merged and the distances are recalculated. To minimize the amount of necessary interactions with the user, *WannaDB* does not always ask for the pair of clusters with the lowest distance, but chooses a pair with a higher distance, using a step size that is adapted based on the last interactions. If the user confirms the equivalence of the candidates, not only that pair but also those with a substantially lower distances are merged. If the entries of the merging candidates are marked as different, *WannaDB* continues to search for a better threshold for the distance between clusters using a binary search pattern.

## 5 Current Limitations of *WannaDB*

In order to build a system that can quickly compute query results on various domains, we introduce two limitations: First, *WannaDB* currently can only answer single-table queries on top of document collections; i.e., we extract one table per document collection where each row of the table corresponds to one document. However, this is not a severe limitation, since the extracted table can be seen as the materialized result of a join. *WannaDB* will extract a wide table (e.g., containing information about an incident itself but also the airlines and airports involved)—but only with the attributes that are required for a given query.

Second, the results produced by *WannaDB* are always approximate. While *WannaDB* can achieve a high F1-score for all attributes (as we will show below), query results might be incomplete (i.e., values of attributes might be missing) or the extracted values might be dirty (e.g., a group-by statement might result in two instead of one group due to a not fully correct clustering). However, we believe that the query results of *WannaDB* are still of high value to users, providing them with a trend and allowing them to decide if something interesting is contained in the document collection in a short time.

## 6 Experimental Evaluation

In this evaluation, we aim to show the abilities of *WannaDB* on text collections from different domains. We will demonstrate the end-to-end performance, compare our table filling approach to non-interactive and learned models, and evaluate the effects of interaction, and the scalability of *WannaDB*. To the best of our knowledge, there is no system working like *WannaDB* yet. Therefore, we cannot compare our results end-to-end with existing systems. As the whole task of running SQL queries over text collections is quite complex, there is no simple baseline for comparison either. However, we evaluate the components of our approach individually, and show that *WannaDB* performs better compared to various baselines. We perform our evaluation on three data sets from very different domains. Each of them consists of a document collection as well as a ground-truth extraction of structured data that we can use to evaluate the results of executing ad-hoc queries with *WannaDB*.

**Aviation.** The first data set is based on aviation accident reports published by the United States National Transportation Safety Board (NTSB).<sup>6</sup> Each report documents a severe aviation accident and provides details like the prevailing circumstances, probable causes, conclusions, and recommendations. For the experiments, we use the executive summaries that the NTSB publishes with each report. As a ground-truth, we compiled a list of twelve attributes based on frequently occurring facts from the summaries. We then manually created annotations that capture where the summaries mention the attributes' values. The final data set comprises 100 annotated documents and a table which provides the ground-truth structured data for all attributes.

---

<sup>6</sup><https://www.ntsb.gov/investigations/AccidentReports/Pages/Reports.aspx?mode=Aviation>

**COVID19.** The second data set is based on the German RKI’s daily reports outlining the situation of the Covid-19 pandemic in Germany.<sup>7</sup> We again used the summaries of the full documents, which contain information like the number of new laboratory-confirmed Covid-19 cases or the number of Covid-19 patients in intensive care. We compiled a list of seven all-numeric attributes, which is in particular challenging compared to string-valued attributes, since these are harder to separate into different attributes in the embedding space. As a ground-truth for the experiments, we manually annotated the occurrences of all these seven attributes again in 100 reports.

**T-REx: Countries, Nobel & Skyscrapers.** In addition to the data sets before that we explicitly created for evaluating *WannaDB*, we adapted the T-REx data set [E118] that was also used in other papers. The original data set consists of 11 million Wikidata triples aligned with 3.09 million Wikipedia abstracts. We extracted three subsets based on article categories from different domains: *Countries* consists of 187 documents with three annotated attributes, *Nobel* challenges to extract four attributes (date of birth and death, field of work and country) for 209 Nobel Prize laureates, and *Skyscrapers* is by far the largest data set with 2683 documents containing annotations for three attributes. All these data sets are quite sparse, since most of the time only a subset of the attributes is contained in a document. Therefore, this data set is valuable to test how well *WannaDB* can work when information in documents is missing.

**Metrics.** As a main metric, we report the F1 score in most experiments (values between 0 and 1, higher is better) as an aggregated value that incorporates both the precision (i.e., the correctness of the table cell values) of our approach and its recall (i.e., the extent to which table cells are filled as expected). The F1 scores we report are calculated based on the ground truth and predictions in the filled tables. We thereby consider cells (i.e., an attribute value) as true positives when they are correctly filled with information from the text corresponding to that row, and as true negatives when they are correctly left empty, in case the required information is not present in the corresponding text. False positive predictions occur, when a cell is filled incorrectly. False negatives occur when a cell is left empty that should have been filled with data from the text, and also for incorrectly filled cells, as the correct nugget has not been found.

## 6.1 Exp. 1 – End-to-end Queries

To provide an indication of how *WannaDB* works end-to-end, we perform a qualitative analysis on queries involving aggregation and grouping over multiple documents before we later-on show quantitative results for *WannaDB*. For the experiments, we assume that a user always provides correct feedback for *WannaDB* to execute the matching of extractions to query attributes. However, we do not expect optimal feedback, i.e., the simulated feedback actions are not chosen in a way to maximize speed of convergence. We report the results after

---

<sup>7</sup>[https://www.rki.de/DE/Content/InfAZ/N/Neuartiges\\_Coronavirus/Situationsberichte/Gesamt.html](https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Gesamt.html)

using 20 simulated user interactions (i.e., 20 times confirming an extraction or choosing an alternative one as a match for a query attribute). We discuss the interaction effort that is needed for *WannaDB* to perform extractions in a separate experiment.

Figure 5 shows the first five rows of the query results for two aggregation queries executed on the *T-REx Nobel* and the *T-REx Countries* data sets. Additionally, precision and recall, as well as a numeric score of the correctness of the clusters, can be seen. While *WannaDB* delivered the correct values for the group-by operation, the aggregation (COUNT) deviates slightly from the ground-truth. The reason is that for some documents, *WannaDB* could not extract the requested information. As such, the results of *WannaDB* can be seen as an approximation of the true query result that can be used for quickly gaining (initial) insights into text collections. Moreover, it is important to note that existing extraction baselines—that in contrast to *WannaDB* do not support ad-hoc queries—also do not provide perfect extractions (as we show in the following experiments).

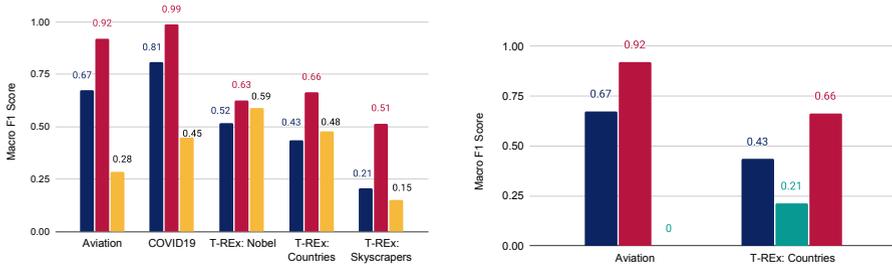
| T-REx nobel laureate: |          | country  | count (correct) | T-REx country: |           | continent     | count (correct) |
|-----------------------|----------|----------|-----------------|----------------|-----------|---------------|-----------------|
| SELECT                | country  | American | 88 (84)         | SELECT         | continent | Africa        | 33 (34)         |
| COUNT(*)              | AS count | Soviet   | 9 (10)          | COUNT(*)       | AS count  | Europe        | 19 (20)         |
| GROUP BY              | country  | Swiss    | 5 (4)           | GROUP BY       | language  | Asia          | 12 (14)         |
| SORT BY               | count    | Japan    | 3               | SORT BY        | count     | South America | 11 (13)         |
| P:                    | 1.0000   | Germany  | 2 (3)           | P:             | 0.7500    | North America | 4 (5)           |
| R:                    | 0.5714   | ...      | ...             | R:             | 1.0000    | ...           | ...             |
| MJI:                  | 0.4775   |          |                 | MJI:           | 0.5004    |               |                 |

Fig. 5: End-to-end results for two queries executed on T-REx data sets. The tables show the first five rows of the resulting table (one attribute column filled by *WannaDB* plus aggregation results). The bracketed values indicate the ground truth values. Additionally, precision (P) and recall (R) computed at cluster level, and mean Jaccard Index (MJI) averaged over all clusters are reported.

## 6.2 Exp. 2 – Interactive Table Extraction

In the second experiment, we quantitatively evaluate how well *WannaDB* can fill a table specified by a user’s query with information from the texts. For this, we focus on the quality of the interactive table extraction, which is the most important step for *WannaDB* to provide high-quality query results; i.e., if the table extraction is not able to provide high accuracy, grouping and filtering will also not be able to provide high accuracy. For showing the quality of *WannaDB*, we run the experiments in this section on all three data sets (Aviation, COVID19 and T-REx).

**Baselines.** To put the results of *WannaDB* into perspective, we compare it to two baselines based on BART [Le20b]. BART is a state-of-the-art pre-trained transformer model, with a high capacity to learn text-based tasks with minimal overhead of fine-tuning. Its robust architecture outperformed older transformers, especially on tasks like question answering. We use the openly available *bart-large* model from the Huggingface [Wo19] library and formulate information extraction for individual query attributes as a sequence-to-sequence



(a) Table filling results. It can be seen that *WannaDB* performs comparable to the BART model trained explicitly on the data (fine-tuned per topic individually) and outperforms the generic BART model (fine-tuned on SQuAD) most of the time.

(b) Upper Baseline BART models show low generalization abilities on unseen data sets—the performance of a BART model trained on one data set drops drastically when applied to the other data set. *WannaDB* for comparison.

Fig. 6: Text-to-Table Results

task (i.e., the input is a text document and the output is the structured data extracted from the text). For fine-tuning BART for the information extraction task on a particular data set (i.e., transforming a text into a table) we use the following procedure: We split each data set into 75% that we use as train set for fine-tuning, 15% as validation set and 10% as a holdout test set. We then fine-tune one BART model on each data set for 50 epochs with a learning rate of  $1e-5$  and batch size of 2, which yielded the best performance in our experiments. Moreover, we select the best checkpoint from the 50 epochs based on the validation set for evaluation. Important to note here is that the resulting fine-tuned BART models are an upper baseline for *WannaDB*, as they are trained supervised on the annotated data and all possible query attributes; i.e., with this baseline we do not test the ad-hoc scenario that we envision for *WannaDB*, but instead assume that all query attributes are known in advance.

For comparing *WannaDB* to a baseline that supports ad-hoc queries on a new (unseen) text collection, we use a second variant that is also based on BART but not pre-trained on the particular data set and query attributes. For this baseline, we instead use a BART model<sup>8</sup> that is already fine-tuned for extracting structured information from the SQuAD 2.0 data set [RJL18].<sup>9</sup> For the experiment, we use this fine-tuned model on an unseen data set and extract attributes that the model has not seen during fine-tuning.

**WannaDB vs. Baselines.** The results of *WannaDB* in comparison with the two BART models are shown in Figure 6a. For *WannaDB*, we report the median over 20 randomized runs, and again use 20 simulated user interactions per attribute. As baselines, we use the two variants of BART discussed before.<sup>10</sup> BART models fine-tuned per data set (red bars)

<sup>8</sup>Used Checkpoint: *phiyodr/bart-large-finetuned-squad2* from Huggingface [Wo19]

<sup>9</sup>In particular the fine-tuning task is QA on text collections which can be used to extract query attributes.

<sup>10</sup>The results of *WannaDB* and the second BART model that is used out-of-the-box are calculated on the whole data sets, whereas the results of the first BART model that is fine-tuned for the given data set are computed only on the 10% holdout test sets.

are able to achieve high F1 scores on the data and query attributes they were trained on, outperforming *WannaDB* on all data sets. Nevertheless, this approach is relying on the availability of annotated training data, which prevents ad-hoc queries. In comparison to the BART model that is used without fine-tuning on a given data set and set of query attributes (yellow bar), *WannaDB* achieves substantially better results. Especially for the Aviation and COVID19 data sets, *WannaDB* clearly outperforms this BART baseline. On the T-REx data sets, *WannaDB* provides competitive or better performance depending on the subset of data. We assume that BART’s performance on the T-REx data is influenced by the fact that both the SQuAD data set it was fine-tuned on and the T-REx data set are based on Wikipedia.

**Generalization of BART.** As we have seen, while fine-tuning a BART model per data set yields the best performance, the BART model that is not fine-tuned for a data set provides inferior performance up to a point that it cannot extract any attributes correctly. To understand the generalization capabilities of BART in more depth and see if this is a systematic problem of BART, we now systematically use BART on data sets it has not been fine-tuned for. To be more precise, Figure 6b shows the results of two fine-tuned BART models: one fine-tuned on the *Aviation* data set and then used on the *T-REx Countries* data set and another model that we used vice versa; i.e., we applied both of them to the respective other data set, for which they have not been fine-tuned. The model fine-tuned on the *Aviation data* (reaching an F1 score of 91.95% tested in-domain on the *Aviation* data) only achieves 21.23% when tested on the *T-REx Countries* data set. At the same time, the model fine-tuned on the *T-REx Countries* data set (reaching an F1 score of 0.6633 on the in-domain test set) fails completely for extracting information correctly from the unseen aviation data domain with an F1-score of 0.0. This shows that a fine-tuned BART model is a valid approach to information extraction when annotated data is available and a fixed set of attributes is queried, but the resulting models are not able to generalize ad-hoc to other domains. In contrast, the results of *WannaDB* show that it can generalize well across data sets even without any particular training per data set and that the interactive approach provides an advantage over using generic embeddings or transformers directly.

**Detailed Analysis of *WannaDB*.** As a last point, we now zoom into the performance of *WannaDB* and analyze the results for all data sets on a per-attribute level to show that *WannaDB* can provide stable high performance and not just high performance for some query attributes. We used a combination of two different named entity recognizers,<sup>11</sup> Stanza [Qi20] and SpaCy<sup>12</sup> [Ho20] followed by our interactive matching approach.

Figure 7 shows that *WannaDB* can provide high accuracy and recall (measured by the combining F1 score, blue bars, right axis) for a wide spectrum of attributes from the three different data sets used in our evaluation. However, for some attributes the table is filled with a much lower quality than for others or not at all (e.g., for weather conditions). One

<sup>11</sup>*WannaDB* allows using multiple extractors at the same time, even if they produce overlapping nuggets. As default configuration for *WannaDB* and our experiments, we employ a combination of two robust general purpose extractors that are designed to work for a broad variety of domains. However, any other (combination of) extractors could be used in *WannaDB* as well.

<sup>12</sup>Using the *en\_core\_web\_lg* model

reason can be that the currently employed information extractors are not able to extract the necessary information nuggets from the text (yellow bars). In particular, *aircraft\_damage* and *weather\_condition* are examples, where not only a large heterogeneity of mentions can be found but also very domain-specific terminology is used. Another reason for low table filling quality can be that the attributes occur in only a small fraction of the documents, as in the case of the attribute *owned\_by* (which only occurs in 6% of the documents).

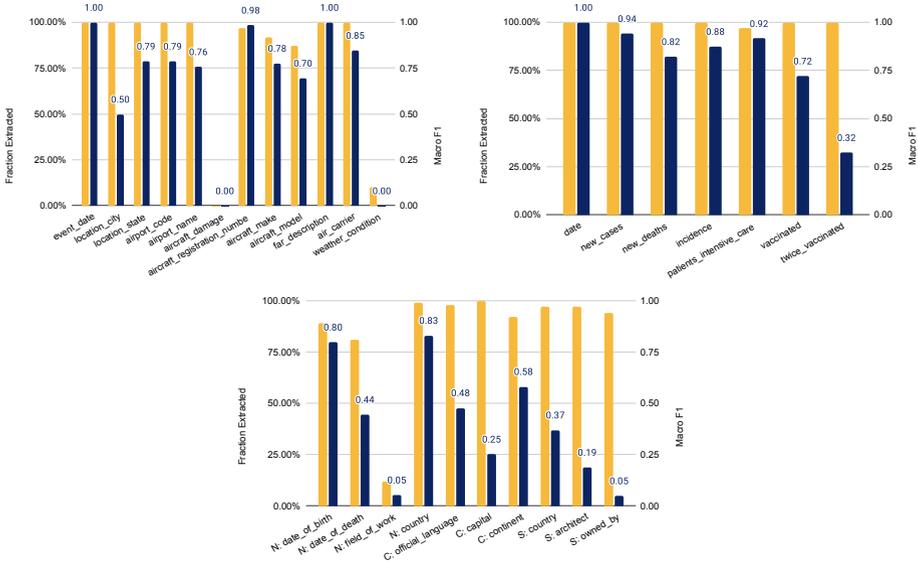


Fig. 7: ■ Fraction of values that could be extracted successfully and ■ table filling results per attribute of the Aviation, COVID19 and T-REx data sets (in this order). *WannaDB* produces high scores for the majority of attributes, more than half are 0.7 or above.

In conclusion, *WannaDB* has the advantage over fine-tuned BART models, that it neither requires annotated training data, nor several hours of training time in order to work on unseen text collections. Furthermore, it does not suffer from the problem of *hallucination* [Ma20] that transformer-like models regularly experience, since they aim to also generate values for attributes even if no information nugget is present in the text. *WannaDB* instead generates an empty value in that case.

### 6.3 Exp. 3 – Effects of Interaction

In the previous experiments, we assumed a fixed amount of user interaction. In the third part of our evaluation, we instead investigate how the amount of interactive feedback given affects the table filling performance of *WannaDB*. We therefore simulate the interactive matching process with different interaction limits (i.e., the number of interactions per extracted query attribute). The resulting F1 scores can be seen in Figure 8.

As we can see, for some attributes, *WannaDB* achieves very high F1 scores with only one interaction with the user (e.g., for *event date* or *aircraft registration number* in the *Aviation* data set). These are attributes where the entity type of the extracted information nugget is very similar to the attribute name or the pattern of the extracted information nugget is rather unique. For example, the extraction has the named entity tag *DATE* which is similar to *event date*. For other attributes though, the performance of *WannaDB* strongly depends on the amount of interactive feedback. However, important is that *WannaDB* can typically provide high quality with only a few interactions. For most attributes, the first 5 – 10 interactions massively improve the F1-score to achieve gains of up to 0.5. This overall confirms the interactive matching procedures we presented in Section 4 and the algorithm to select the right threshold. Yet, as we can additionally see, for a few attributes (e.g., weather condition), even many interactions cannot further improve the F1 scores. As we showed in the last experiment, the reason is that none of the extractors used in *WannaDB* can provide the information nugget for this attribute. Thus, as a future direction we want to combine *WannaDB* with a much broader set of existing extraction approaches beyond the named entity recognizers which we currently use, such as approaches for open information extraction.

#### 6.4 Exp. 4 – Scalability

In our final experiment, we aim to assess the scalability of *WannaDB* to large text collections. Since *WannaDB* is an interactive system, the response times experienced by users are the most important performance metric. Across all used data sets, we measure that *WannaDB* takes on average 0.43 seconds to process a single user interaction.<sup>13</sup> This latency includes all computations between two user interactions; i.e., updating the cached distances and guessed matches as well as presenting the next set of candidate matches to the user for feedback. In general, we find that the interaction latency scales linearly with the number of nuggets. To measure the offline extraction phase, which has to be executed only once per text collection, we report the runtime on our largest data set *T-REx Skyscrapers*, which comprises 2,683 documents. Running our default extraction phase takes about 48 minutes and produces 102,467 nuggets. Comparing runtimes across data sets, we again find that the extraction runtime scales linearly with the number of generated nuggets.

In summary, it can be seen that *WannaDB* can scale to extensive text collections with thousands of documents and more than 100,000 information nuggets by finishing the offline phase in a reasonable time and providing response times that allow for an interactive usage of the system [LH14].

---

<sup>13</sup>We executed this and all other of our experiments on a consumer desktop machine (CPU: AMD Ryzen 9 3900X; RAM: 32GB @3000MHz; GPU: NVIDIA GeForce RTX 2070 SUPER with 8GB VRAM).

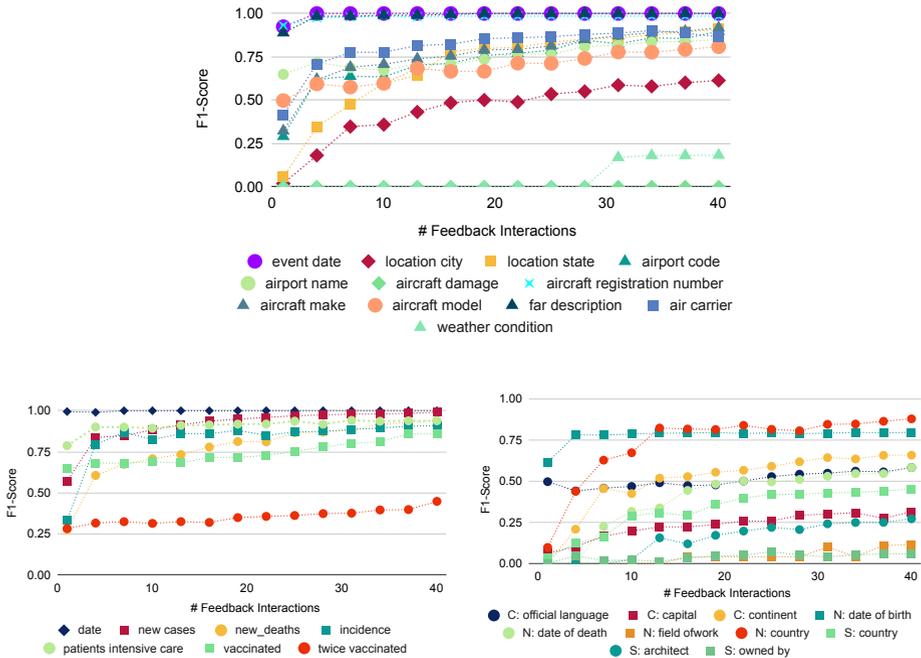


Fig. 8: F1 scores of *WannaDB* for the different attributes of the *Aviation*, *COVID19* and *T-REX* data sets for different amounts of feedback iterations per attribute (1-40). For most attributes, already a small amount of interactions drastically improves the quality, and more interactions lead to continuous improvements.

## 7 Related Work

Running SQL queries on text collections is a new task, and to the best of our knowledge, there is no other system yet working in the same way as *WannaDB*. However, some parts of the task resemble existing tasks and for some components of our approach there is previous work. Therefore, in this section, we give an overview of the related work of different areas, including knowledge base population and schema matching based on embeddings.

**Information Extraction Systems.** Existing approaches to answer queries over text collections heavily rely on manual labor, requiring users either to read through vast amounts of texts and extract relevant information manually, or to build specific extraction pipelines. One category of information extraction systems focuses on the task of knowledge base population, where a graph-structured knowledge base is constructed or expanded based on knowledge from natural language texts. Extractive approaches like DeepDive [Sa16], SystemT [Ch10], DefIE [BTN15], and QKBFly [Ng17] build upon (open) information extractors like ClausIE [CG13] and also perform the adaption, cleaning, and combination

stages of the knowledge base building process. Most of these approaches require high manual efforts to design extraction pipelines for each knowledge base and domain specifically. Google Squared could be used to create fact-tables similar to the ones we propose from web contents, but was unfortunately discontinued without publications about the underlying techniques. Closest to our work are recent approaches for query-driven on-the-fly knowledge base construction, such as QKBFly. Yet, QKBFly extracts general subject-predicate-object triples and does not populate a user-defined table as *WannaDB* does. The vision of INODE [Am21] is to provide an end-to-end data exploration system that is also able to include information from natural language texts. For this task, the knowledge base population approach LILLIE [Sm22] extracts triples from text domain-independently. However, the system has not been thoroughly evaluated for generalization to unseen domains. Recent approaches use transformer models to tackle information extraction tasks like relation extraction [EU21, CN21, Ng20] in an end-to-end fashion to avoid the errors accumulating in pipeline-based approaches. However, transformer-based methods are costly to train and suffer from issues like hallucination [Ma20]. A more explainable approach to information extraction is introduced by [Ko22, Re21] with a framework for learning text classifiers with a human-in-the-loop. Recently, [Sa22] introduced an interactive system that allows users to specify templates that are then used to perform zero-shot information extraction.

**Text-To-Table.** The idea of automatically transforming a text into a table was also approached by [WZL22] as text-to-table task, which inversely tackles the well studied table-to-text problem. Yet, their work is not directly comparable, since they assume that each text fills one or more entire tables, while we assume that a text collection fills one table in which each text corresponds to a row.

**Template Filling & Named Entity Recognition.** The goal of slot or template filling is similar to our objective [GS96], yet in contrast to our approach, most template filling approaches are specifically crafted for a fixed set of slots. A common approach to extract a fixed set of attributes from a text is to learn a named entity recognizer specifically for the desired entity types (e.g., [SJ19]). Named entity recognizers extract a set of entity types like organizations, locations, or products from natural language texts. However, the training requires a substantial amount of annotated data, and the learned system will not generalize to entity types not present in the training data. Some approaches (e.g., [Ch15, We19, Kh17]) attempt to avoid this problem by using active learning, which allows the learning algorithm to query the user, for example by selecting training instances that the user then labels by hand. Another strategy is distantly-supervised or weakly-supervised named entity recognition (e.g., [Fr17, Li20]). In contrast to our system, these approaches train named entity recognizers specifically for the desired set of entity types, whereas we use the output of conventional named entity recognizers to populate the user-provided attributes. Together with the interactive matching, this allows *WannaDB* to generalize to unseen domains without the costly training of domain-specific named entity recognizers.

**Other Matching Tasks.** Approaches for schema matching (e.g., [Hä20, He20]), are related to *WannaDB*, too, since we frame the mapping between the information extractors' output

and the user-provided list of attributes as a matching problem, but try to find correspondences between attributes and possible values, and not between columns or even full tables. Another recent approach focuses on matching texts to structured data, in particular also matching texts to table rows [ASP21]. Yet, this task differs from the matching task in *WannaDB*, as it assumes the tables are given, whereas in *WannaDB* a table is filled through the matching.

**Entity Disambiguation & Cross Document Co-Reference Resolution.** The surface form of an entity in a text is often not sufficient to uniquely identify it. Yet, knowing whether two mentions of the same type describe the same entity is relevant for correct grouping in our case, but also existing tasks like entity linking/knowledge base alignment. For the latter there are three main challenges (see [Dr10]): name variations (e.g., different mention forms, abbreviations, alternate spellings, and aliases), entity ambiguity (same written form for different entities), and absence (i.e., the text mentions a previously unknown entity). The last one is not relevant for our use-case, since we do not rely on a given KB but build tables only based on the current text collection. We can concentrate on the problem of ambiguity, i.e., decide, whether two nuggets that were matched as different rows of the same attribute are in fact the same or represent different concepts. The field of computing equivalence classes of textual mentions for the same entity is called cross-document co-reference resolution (CCR). It was, e.g., tackled by [DW15, KCP18, Ca21], but these existing approaches often concentrate only on entities from certain domains or of certain types (like events).

**Prior Results of *WannaDB*.** A first version of the matching component of *WannaDB* including an initial evaluation on two real-world data sets was published at [HBB21]. In this paper, we pick up the vision of the whole application cycle presented at [Hä21]. As such, we present the integration of the table extraction procedure of *WannaDB* into a full system. Moreover, compared to the original submission, we also developed a new interactive matching procedure where we leverage the human ability to quickly find patterns by presenting multiple guessed matches at once, which allows users to quickly correct wrong matches. Multiple ways to give feedback (confirm, fix, or mark that there is no match in the document) further enhance quality and flexibility of matching. A demo of the interactive GUI for this matching process was presented at [HBB22].

## 8 Conclusions

In this paper, we presented *WannaDB*, a novel tool to explore the contents of unstructured data (text) using SQL-like queries in an ad-hoc fashion and without the need to manually design extraction pipelines upfront. It builds on embeddings and a novel interactive query execution strategy and consists of components to infer the required table structure from the query, extract and organize the required information from the text, group results on the embedding level and execute the query. Our evaluation shows that the individual components of *WannaDB* can achieve similar performance to models trained on large data sets for partial or related tasks, and gives an impression of the end-to-end quality that makes *WannaDB* suitable for many exploratory use cases.

## Acknowledgments

This work has been supported by the German Federal Ministry of Education and Research (BMBF) as part of the Project Software Campus 2.0 under grant ZN 01IS17050, by the BMBF and the state of Hesse as part of the NHR Program, the Hochtief project AICO (AI in Construction), and the HMWK cluster project 3AI. Finally, we want to thank hessian.AI, and the Centre Responsible Digitality (ZEVEDI) at TU Darmstadt, as well as DFKI Darmstadt for their support.

## Bibliography

- [Am21] Amer-Yahia, Sihem; Koutrika, Georgia; Braschler, Martin; Calvanese, Diego; Lanti, Davide; Lücke-Tieke, Hendrik; Mosca, Alessandro; de Farias, Tarcisio Mendes; Papadopoulos, Dimitris; Patil, Yogendra; Rull, Guillem; Smith, Ellery; Skoutas, Dimitrios; Subramanian, Srividya; Stockinger, Kurt: INODE: Building an End-to-End Data Exploration System in Practice. *SIGMOD Rec.*, 50(4):23–29, 2021.
- [ASP21] Ahmadi, Naser; Sand, Hansjorg; Papotti, Paolo: Unsupervised Matching of Data and Text. *CoRR*, abs/2112.08776, 2021.
- [BTN15] Bovi, Claudio Delli; Telesca, Luca; Navigli, Roberto: Large-Scale Information Extraction from Textual Definitions through Deep Syntactic and Semantic Analysis. *Trans. Assoc. Comput. Linguistics*, 3:529–543, 2015.
- [Ca21] Cattan, Arie; Johnson, Sophie; Weld, Daniel S.; Dagan, Ido; Beltagy, Iz; Downey, Doug; Hope, Tom: SciCo: Hierarchical Cross-Document Coreference for Scientific Concepts. In: 3rd Conference on Automated Knowledge Base Construction, AKBC 2021, Virtual, October 4-8, 2021. 2021.
- [CG13] Corro, Luciano Del; Gemulla, Rainer: ClausIE: clause-based open information extraction. In: 22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013. International World Wide Web Conferences Steering Committee / ACM, pp. 355–366, 2013.
- [Ch10] Chiticariu, Laura; Krishnamurthy, Rajasekar; Li, Yunyao; Raghavan, Sriram; Reiss, Frederick; Vaithyanathan, Shivakumar: SystemT: An Algebraic Approach to Declarative Information Extraction. In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, July 11-16, 2010, Uppsala, Sweden. The Association for Computer Linguistics, pp. 128–137, 2010.
- [Ch15] Chen, Yukun; Lasko, Thomas A.; Mei, Qiaozhu; Denny, Joshua C.; Xu, Hua: A study of active learning methods for named entity recognition in clinical text. *J. Biomed. Informatics*, 58:11–18, 2015.
- [CN21] Cabot, Pere-Lluís Huguet; Navigli, Roberto: REBEL: Relation Extraction By End-to-end Language generation. In: *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*. Association for Computational Linguistics, pp. 2370–2381, 2021.

- [De19] Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). Association for Computational Linguistics, pp. 4171–4186, 2019.
- [Dr10] Dredze, Mark; McNamee, Paul; Rao, Delip; Gerber, Adam; Finin, Tim: Entity Disambiguation for Knowledge Base Population. In: COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China. Tsinghua University Press, pp. 277–285, 2010.
- [DW15] Dutta, Sourav; Weikum, Gerhard: Cross-Document Co-Reference Resolution using Sample-Based Clustering with Knowledge Enrichment. *Trans. Assoc. Comput. Linguistics*, 3:15–28, 2015.
- [El18] ElSahar, Hady; Vougiouklis, Pavlos; Remaci, Arslan; Gravier, Christophe; Hare, Jonathon S.; Laforest, Frédérique; Simperl, Elena: T-REx: A Large Scale Alignment of Natural Language with Knowledge Base Triples. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018. European Language Resources Association (ELRA), 2018.
- [EU21] Eberts, Markus; Ulges, Adrian: An End-to-end Model for Entity-level Relation Extraction using Multi-instance Learning. In: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021. Association for Computational Linguistics, pp. 3650–3660, 2021.
- [Fr17] Fries, Jason A.; Wu, Sen; Ratner, Alexander; Ré, Christopher: SwellShark: A Generative Model for Biomedical Named Entity Recognition without Labeled Data. *CoRR*, abs/1704.06360, 2017.
- [GS96] Grishman, Ralph; Sundheim, Beth: Message Understanding Conference - 6: A Brief History. In: 16th International Conference on Computational Linguistics, Proceedings of the Conference, COLING 1996, Center for Sprogteknologi, Copenhagen, Denmark, August 5-9, 1996. pp. 466–471, 1996.
- [Hä20] Hättasch, Benjamin; Truong-Ngoc, Michael; Schmidt, Andreas; Binnig, Carsten: It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings. In: 2nd International Workshop on Applied AI for Database Systems and Applications (AIDB20). In conjunction with the 46th International Conference on Very Large Data Bases, Virtual, August 31 - September 4, 2020. 2020.
- [Hä21] Hättasch, Benjamin: WannaDB: Ad-hoc Structured Exploration of Text Collections Using Queries. In: Proceedings of the Second International Conference on Design of Experimental Search Information REtrieval Systems, Padova, Italy, September 15-18, 2021. volume 2950 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 179–180, 2021.
- [HBB21] Hättasch, Benjamin; Bodensohn, Jan-Micha; Binnig, Carsten: ASET: Ad-hoc Structured Exploration of Text Collections. In: 3rd International Workshop on Applied AI for Database Systems and Applications (AIDB21). In conjunction with the 47th International Conference on Very Large Data Bases, Copenhagen, Denmark, August 16 - 20, 2021. 2021.

- [HBB22] Hättasch, Benjamin; Bodensohn, Jan-Micha; Binnig, Carsten: Demonstrating ASET: Ad-hoc Structured Exploration of Text Collections. In: SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022. ACM, pp. 2393–2396, 2022.
- [He20] Hernández, Daniel Ayala; Hernández, Inma; Ruiz, David; Rahm, Erhard: LEAPME: Learning-based Property Matching with Embeddings. CoRR, abs/2010.01951, 2020.
- [He21] Hendrycks, Dan; Burns, Collin; Kadavath, Saurav; Arora, Akul; Basart, Steven; Tang, Eric; Song, Dawn; Steinhardt, Jacob: Measuring Mathematical Problem Solving With the MATH Dataset. In: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual. 2021.
- [Ho20] Honnibal, Matthew; Montani, Ines; Van Landeghem, Sofie; Boyd, Adriane: spaCy: Industrial-strength Natural Language Processing in Python. 2020.
- [KCP18] Kenyon-Dean, Kian; Cheung, Jackie Chi Kit; Precup, Doina: Resolving Event Coreference with Supervised Representation Learning and Clustering-Oriented Regularization. In: Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, \*SEM@NAACL-HLT 2018, New Orleans, Louisiana, USA, June 5-6, 2018. Association for Computational Linguistics, pp. 1–10, 2018.
- [Kh17] Kholghi, Mahnoosh; Vine, Lance De; Sitbon, Laurianne; Zuccon, Guido; Nguyen, Anthony N.: Clinical information extraction using small data: An active learning approach based on sequence representations and word embeddings. *J. Assoc. Inf. Sci. Technol.*, 68(11):2543–2556, 2017.
- [Ko22] Kovács, Ádám; Gémes, Kinga; Iklódi, Eszter; Recski, Gábor: POTATO: exPlainable infOrmation exTrAcTion framewOrk. CoRR, abs/2201.13230, 2022.
- [Le20a] Lembo, Domenico; Li, Yunyao; Popa, Lucian; Scafoglieri, Federico Maria: Ontology mediated information extraction in financial domain with Mastro System-T. In: Proceedings of the Sixth International Workshop on Data Science for Macro-Modeling, DSMM 2020, In conjunction with the ACM SIGMOD/PODS Conference, Portland, OR, USA, June 14, 2020. ACM, pp. 3:1–3:6, 2020.
- [Le20b] Lewis, Mike; Liu, Yinhan; Goyal, Naman; Ghazvininejad, Marjan; Mohamed, Abdelrahman; Levy, Omer; Stoyanov, Veselin; Zettlemoyer, Luke: BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020. Association for Computational Linguistics, pp. 7871–7880, 2020.
- [LH14] Liu, Zhicheng; Heer, Jeffrey: The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2122–2131, 2014.
- [Li20] Liang, Chen; Yu, Yue; Jiang, Haoming; Er, Siawpeng; Wang, Ruijia; Zhao, Tuo; Zhang, Chao: BOND: BERT-Assisted Open-Domain Named Entity Recognition with Distant Supervision. In: KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020. ACM, pp. 1054–1064, 2020.

- [Ma14] Manning, Christopher D.; Surdeanu, Mihai; Bauer, John; Finkel, Jenny Rose; Bethard, Steven; McClosky, David: The Stanford CoreNLP Natural Language Processing Toolkit. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations. The Association for Computer Linguistics, pp. 55–60, 2014.
- [Ma20] Maynez, Joshua; Narayan, Shashi; Bohnet, Bernd; McDonald, Ryan T.: On Faithfulness and Factuality in Abstractive Summarization. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020. Association for Computational Linguistics, pp. 1906–1919, 2020.
- [Mi18] Mikolov, Tomáš; Grave, Edouard; Bojanowski, Piotr; Puhersch, Christian; Joulin, Armand: Advances in Pre-Training Distributed Word Representations. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018. European Language Resources Association (ELRA), 2018.
- [Ng17] Nguyen, Dat Ba; Abujabal, Abdalghani; Tran, Khanh; Theobald, Martin; Weikum, Gerhard: Query-Driven On-The-Fly Knowledge Base Construction. Proc. VLDB Endow., 11(1):66–79, 2017.
- [Ng20] Nguyen, Minh-Tien; Le, Dung Tien; Son, Nguyen Hong; Minh, Bui Cong; Duong, Do Hoang Thai; Linh, Le Thai: Understanding Transformers for Information Extraction with Limited Data. In: Proceedings of the 34th Pacific Asia Conference on Language, Information and Computation, PACLIC 2020, Hanoi, Vietnam, October 24-26, 2020. Association for Computational Linguistics, pp. 478–487, 2020.
- [Qi20] Qi, Peng; Zhang, Yuhao; Zhang, Yuhui; Bolton, Jason; Manning, Christopher D.: Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2020, Online, July 5-10, 2020. Association for Computational Linguistics, pp. 101–108, 2020.
- [Re21] Recski, Gábor; Lellmann, Björn; Kovács, Ádám; Hanbury, Allan: Explainable Rule Extraction via Semantic Graphs. In: Joint Proceedings of the Workshops on Automated Semantic Analysis of Information in Legal Text (ASAIL 2021) and AI and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA 2021) held online in conjunction with 18th International Conference on Artificial Intelligence and Law (ICAAIL 2021). volume 2888 of CEUR Workshop Proceedings, CEUR-WS.org, Sao Paolo, Brazil (held online), pp. 24–35, 2021.
- [RG19] Reimers, Nils; Gurevych, Iryna: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019. Association for Computational Linguistics, pp. 3980–3990, 2019.
- [RJL18] Rajpurkar, Pranav; Jia, Robin; Liang, Percy: Know What You Don’t Know: Unanswerable Questions for SQuAD. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers. Association for Computational Linguistics, pp. 784–789, 2018.

- [Sa16] Sa, Christopher De; Ratner, Alexander; Ré, Christopher; Shin, Jaeho; Wang, Feiran; Wu, Sen; Zhang, Ce: DeepDive: Declarative Knowledge Base Construction. *SIGMOD Rec.*, 45(1):60–67, 2016.
- [Sa22] Sainz, Oscar; Qiu, Haoling; Lopez de Lacalle, Oier; Agirre, Eneko; Min, Bonan: ZS4IE: A toolkit for Zero-Shot Information Extraction with simple Verbalizations. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: System Demonstrations*. Association for Computational Linguistics, Hybrid: Seattle, Washington + Online, pp. 27–38, July 2022.
- [SJ19] Sharma, Shreyas; Jr., Ron Daniel: BioFLAIR: Pretrained Pooled Contextualized Embeddings for Biomedical Sequence Labeling Tasks. *CoRR*, abs/1908.05760, 2019.
- [Sm22] Smith, Ellery; Papadopoulos, Dimitris; Braschler, Martin; Stockinger, Kurt: LILLIE: Information extraction and database integration using linguistics and learning-based algorithms. *Inf. Syst.*, 105:101938, 2022.
- [We19] Wei, Qiang; Chen, Yukun; Salimi, Mandana; Denny, Joshua C.; Mei, Qiaozhu; Lasko, Thomas A.; Chen, Qingxia; Wu, Stephen; Franklin, Amy; Cohen, Trevor; Xu, Hua: Cost-aware active learning for named entity recognition in clinical text. *J. Am. Medical Informatics Assoc.*, 26(11):1314–1322, 2019.
- [Wo19] Wolf, Thomas; Debut, Lysandre; Sanh, Victor; Chaumond, Julien; Delangue, Clement; Moi, Anthony; Cistac, Pierrick; Rault, Tim; Louf, R’emi; Funtowicz, Morgan; Brew, Jamie: HuggingFace’s Transformers: State-of-the-art Natural Language Processing. *ArXiv*, abs/1910.03771, 2019.
- [WZL22] Wu, Xueqing; Zhang, Jiacheng; Li, Hang: Text-to-Table: A New Way of Information Extraction. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2022, Dublin, Ireland, May 22-27, 2022. Association for Computational Linguistics, pp. 2518–2533, 2022.



# NN2SQL: Let SQL Think for Neural Networks

Maximilian E. Schüle,<sup>1</sup> Alfons Kemper,<sup>2</sup> Thomas Neumann<sup>3</sup>



**Abstract:** Although database systems perform well in data access and manipulation, their relational model hinders data scientists from formulating machine learning algorithms in SQL. Nevertheless, we argue that modern database systems perform well for machine learning algorithms expressed in relational algebra. To overcome the barrier of the relational model, this paper shows how to transform data into a relational representation for training neural networks in SQL: We first describe building blocks for data transformation in SQL. Then, we compare an implementation for model training using array data types to the one using a relational representation in SQL-92 only. The evaluation proves the suitability of modern database systems for matrix algebra, although specialised array data types perform better than matrices in relational representation.

**Keywords:** SQL-92, Neural Networks, Automatic Differentiation

## 1 Introduction

Modern database systems generate code to achieve a nearly hard-coded performance. In pipelined processing, code-generation eliminates interpreted function calls, so that the generated machine code processes data in-place of CPU registers. Together with modern hardware trends leading to a performance increase of database servers, code-generation allows database systems to take over more complex computations. One example for complex computations is the emergence of machine learning [Bu22] to solve several tasks such as image classification or even replacing database system's components [He22; MD22]. These tasks rarely happen within database systems but in external tools [Re22; WP22] requiring the data to be extracted from database systems [Na22]. Thus, current research mostly focuses on eliminating the extraction process [Bu20; Ma15; Sc21a; SK22; WGR20] and developing systems that combine data management and machine learning [Ra18]. In contrast, in this paper, we argue that code generation allows database systems to perform well for machine learning when training neural networks [WH21] based on matrix algebra in SQL only [MAF21; OVZ22; Sa22; Sc19; Sc21d].

In a previous study, we stated that training neural networks in SQL is possible as long as the database system provides an array data type and recursive tables for gradient descent [Sc21c]. However, the use of an array as a nested data type interferes with the first normal form (referring to the definition of arrays as a non-atomic data type) and requires copying the data

<sup>1</sup> University of Bamberg, An der Weberei 5, 96047 Bamberg, maximilian.schuele@uni-bamberg.de

<sup>2</sup> TUM, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, kemper@in.tum.de

<sup>3</sup> TUM, Chair for Database Systems, Boltzmannstraße 3, 85748 Garching, neumann@in.tum.de

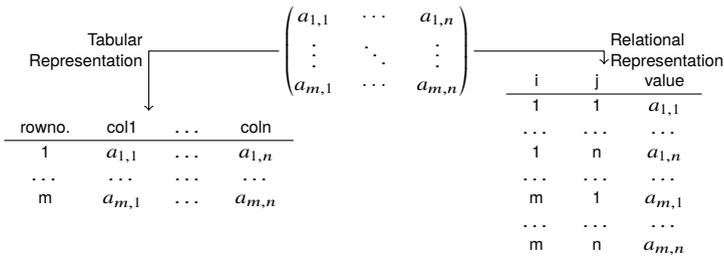


Fig. 1: Tabular and relational representation of matrices in database systems: the latter is used in this study for representing the weights and training neural networks.

between operations. Instead, to process data in-place of CPU registers, we suggested an array backend for code-generating database systems [Sc21b], which stores matrices in a relational representation (cf. Figure 1). This representation stores arrays in normal form with the indices and the elements as table attributes [Sc22]. In a vision paper, Blacher et al. [B122] combined our both approaches to show that recursive CTEs (common table expressions) [DMG22] can deal with matrices in relational representation as input. Nevertheless, their study was limited to logistic regression using matrix algebra and no study has benchmarked training neural networks in SQL without further extensions such as arrays before.

In this paper, we even argue that the relational representation allows database systems to efficiently process the computations along with neural networks. Therefore, this paper uses the relational representation of matrices to train neural networks. We first describe the mathematical background for reverse mode automatic differentiation that is needed to understand the individual matrix operations. We then discuss the intuitive implementation in Python and deduce an implementation in SQL using the relational representation. This includes building blocks for data transformation using one-hot-encoding, matrix/Hadamard product and recursive tables to imitate procedural loops. The evaluation compares the relational representation to the use of array data types within the Umbra database system. An implementation in Python provides the baseline, whose runtime is compared with regard to the batch size and the size of the hidden layer. We conclude with an outlook on optimising recursive tables for this context and on automatically generating the proposed queries.

## 2 Machine Learning in SQL

This section first describes the theoretical background for training neural networks and names the variables, which are later used to name the CTEs. Each variable represents one cached expression computed in the forward pass on function evaluation or in the backward pass on deriving the weight matrices. To discuss the derivation rules, we exemplarily choose a neural network with one hidden layer. Although this limits the number of hidden layers, the derivation rules can be applied similarly to deep neural networks with further weight

---

**Algorithm 1** Automatic Differentiation (Matrices)

---

```

1: function DERIVE( $Z, seed$ )
2:   if  $Z = X + Y$  then DERIVE( $X, seed$ ); DERIVE( $Y, seed$ )
3:   else if  $Z = X \circ Y$  then DERIVE( $X, seed \circ Y$ ); DERIVE( $Y, seed \circ X$ )
4:   else if  $Z = X \cdot Y$  then DERIVE( $X, seed \cdot Y^T$ ); DERIVE( $Y, seed^T \cdot X$ )
5:   else if  $Z = f(X)$  then DERIVE( $X, seed \circ f'(X)$ )
6:   else  $\frac{\partial}{\partial Z} \leftarrow \frac{\partial}{\partial Z} + seed$ 
7:   end if
8: end function

```

---

matrices in-between. Thus, the limitation keeps the example short enough to present the implementations in SQL.

### 2.1 Theoretical Background

Neural networks consist of subsequently applied matrix multiplications each followed by an activation function. They transform an input vector  $x$  with  $m$  attributes into a vector of probabilities for  $l$  categories. With one hidden layer of size  $h$ , we gain two weights matrices  $w_{xh} \in \mathbb{R}^{m \times h}$  and  $w_{ho} \in \mathbb{R}^{h \times l}$ . The first one computes the vector  $a_{xh} \in \mathbb{R}^h$  for the hidden layer, the second one the result vector  $a_{ho} \in \mathbb{R}^l$ . Each activation function returns a normalised value (e.g.  $sig(x) \in [0, 1]$ , Equation 1) that is interpreted as the probability per category. The result vector is compared to the one-hot-encoded categorical label ( $y_{ones}$ ). The difference is elementwisely taken to the power of two ( $\square^2$ ), which is called mean squared error, a common loss function (Equation 3).

$$sig(x) = (1 + e^{-x})^{-1}, \tag{1}$$

$$m_{w_{xh}, w_{ho}}(x) = \underbrace{sig(\overbrace{sig(x \cdot w_{xh})}^{a_{xh}} \cdot w_{ho})}_{a_{ho}}, \tag{2}$$

$$l(x, y_{ones}) = (m_{w_{xh}, w_{ho}}(x) - y_{ones})^{\circ 2}. \tag{3}$$

After computing the loss, reverse mode automatic differentiation computes the derivatives per weight matrix in one pass. This mode derives a function  $f(g(l))$  by decomposing and partially deriving its parts in top-down order:  $\frac{\partial f(g(l))}{\partial l} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial l}$ . Alg. 1 shows reverse mode automatic differentiation for matrices [Mu17]: The function DERIVE takes as input an arithmetic expression  $Z$  and a seed value  $seed$  (the parent partial derivation). The algorithm follows pattern matching on the arithmetic expression  $Z$  to compute and further propagate the partial derivatives until arriving at a leaf node.

By step-wise applying the derivation rules, we obtain the expression tree shown in Figure 2. The derivative of mean squared error calculates the difference between propagated

probabilities and the one-hot-encoded labels (Equation 4). This value gets propagated as initial seed value. Each seed value is elementwise multiplied to each partial derivation, so either the derivation of each activation function (Equation 5, 7) or the matrix multiplication (Equation 6). Finally, the derivation of each weight matrix times the learning rate  $\gamma$  is subtracted from the weight matrix to form the updated weights (Equation 8, 9).

$$l_{ho} = 2 \cdot (m_{w_{xh}, w_{ho}}(x) - y_{ones}), \tag{4}$$

$$\delta_{ho} = l_{ho} \circ \text{sig}'(a_{ho}) = l_{ho} \circ a_{ho} \circ (1 - a_{ho}), \tag{5}$$

$$l_{xh} = \delta_{ho} \cdot w_{ho}^T, \tag{6}$$

$$\delta_{xh} = l_{xh} \circ \text{sig}'(a_{xh}) = l_{xh} \circ a_{xh} \circ (1 - a_{xh}), \tag{7}$$

$$w'_{ho} = w_{ho} - \gamma \cdot a_{xh}^T \cdot \delta_{ho}, \tag{8}$$

$$w'_{xh} = w_{xh} - \gamma \cdot x^T \cdot \delta_{xh}. \tag{9}$$

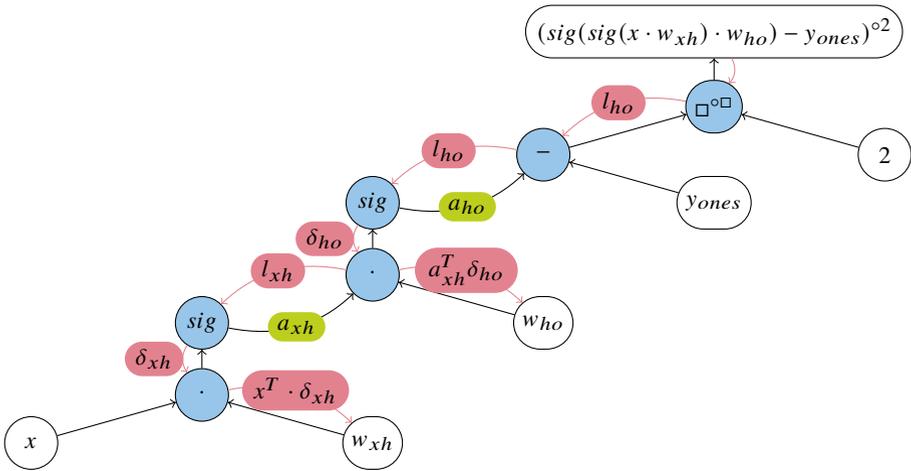


Fig. 2: Automatic differentiation for  $(m_{w_{xh}, w_{ho}}(x) - y_{ones})^2$ .

## 2.2 Implementation in Python and SQL-92

Having defined the equations for training a neural network, we can deduce a Python implementation (List. 1) that uses NumPy for data loading (line 3), transformation (lines 4-8) and generating randomised weights (lines 10-12). Afterwards, a procedural loop (line 14) performs gradient descent that updates the weights according to the derivation rules in each iteration (lines 15-24). So each variable represents one equation needed to backpropagate the loss.

In order to update the weight matrices of neural networks in SQL, we need to map matrix multiplication ( $X \cdot Y$ ), function application ( $f(X)$ ) and elementwise operations

(addition:  $X + Y$ , Hadamard multiplication  $X \circ Y$ ) to the relational representation in SQL. For binary elementwise operations such as Hadamard multiplication or addition/subtraction, a join on the indices combines both tables so that the arithmetic operation is part of the select-clause. Multiplication of two matrices  $m \in \mathbb{R}^{m \times o}$  and  $n \in \mathbb{R}^{o \times n}$  with equal inner dimensions is defined as the sum of the product over  $o$  row/column elements for each entry  $(m \cdot n)_{ij} = \sum_{k=1}^o m_{ik}n_{kj}$ . In relational algebra, this means a join on the inner index, followed by a summation:  $\gamma_{m.i,n.j,sum(m.v \cdot n.v)}(m \bowtie_{m.j=n.i} n)$ . To transpose a matrix in relational representation, only the indices have to be renamed. The corresponding SQL building blocks are shown in List. 3 with their NumPy counterparts in List. 2.

```

1 import numpy as np
2 # load data
3 arr = np.loadtxt("iris.csv", delimiter=",", dtype=float, skiprows=1)
4 X = arr[:,0:4]/10
5 y = arr[:,4].astype(int)
6 # one-hot-encode y
7 y_oh = np.zeros((y.size, y.max()+1))
8 y_oh[np.arange(y.size),y] = 1 # one-hot-encode: set one
9 # initialise weights
10 np.random.seed(1)
11 w_xh = 2*np.random.random((X[0].size,20)) - 1 # size: 4*20
12 w_ho = 2*np.random.random((20,3)) - 1 # size: 20*3
13 # train
14 for j in range(10):
15     print("Iteration:_" + str(j))
16     a_xh = 1/(1+np.exp(-np.dot(X,w_xh))) # sigmoid(x*w_xh)
17     a_ho = 1/(1+np.exp(-np.dot(a_xh,w_ho))) # sigmoid(a_xh*w_ho)
18     l_ho = 2*(a_ho - y_oh)
19     print("Loss:_" + str(np.mean(np.abs(l_ho))))
20     d_ho = l_ho * a_ho * (1-a_ho)
21     l_xh = d_ho.dot(w_ho.T)
22     d_xh = l_xh * a_xh * (1-a_xh)
23     w_ho -= 0.01 * a_xh.T.dot(d_ho)
24     w_xh -= 0.01 * X.T.dot(d_xh)

```

List. 1: Training a neural network with NumPy.

```

1 m.dot(n) # matrix multiplication
2 m * n # hadamard multiplication
3 1/(1+np.exp(-m)) # sigmoid function
4 m.T # transpose

```

List. 2: Building blocks for matrices in NumPy.

```

1 -- create two matrices m and n
2 create table m (i int, j int, v float); create table n (i int, j int, v float);
3 insert into m ...
4 -- matrix multiplication
5 select m.i, n.j, SUM(m.v*n.v) from m inner join n on m.j=n.i group by m.i, n.j
6 -- hadamard multiplication
7 select m.i, m.j, m.v*n.v from m inner join n on m.i=n.i and m.j=n.j
8 -- sigmoid function
9 select i, j, 1/(1+exp(-v)) from m;
10 -- transpose
11 select i as j, j as i, v from m;

```

List. 3: Building blocks for matrices in SQL-92.

| row |              |          |              |          |         | One-Hot-Encoded |     |     |     |     |     | Feature Matrix |
|-----|--------------|----------|--------------|----------|---------|-----------------|-----|-----|-----|-----|-----|----------------|
|     | sepal length | s. width | petal length | p. width | species | i               | j   | v   | i   | j   | v   |                |
| 1   | 5.1          | 3.5      | 1.4          | 0.2      | 0       | 1               | 1   | 1   | 1   | 1   | 5.1 |                |
| ... | ...          | ...      | ...          | ...      | ...     | 1               | 2   | 0   | 1   | 2   | 3.5 |                |
| ... | ...          | ...      | ...          | ...      | ...     | 1               | 3   | 0   | 1   | 3   | 1.4 |                |
| ... | ...          | ...      | ...          | ...      | ...     | ...             | ... | ... | 1   | 4   | 0.2 |                |
| 150 | 5.9          | 3.0      | 5.1          | 1.8      | 2       | 150             | 3   | 1   | ... | ... | ... |                |

Fig. 3: Transformation of the original data set into the relational representation.

To train the neural network in SQL, we first have to convert the data into the relational representation (List. 4). Therefore, we create a table of two indices and a value corresponding to the two-dimensional feature matrix (`img: {[i, j, v]}`, line 3). We assign a column index  $j$  to each attribute of the original input table (lines 5-8) and use the row number as index  $i$ . Afterwards, we one-hot-encode the label: We generate a sparse matrix containing only the one values (line 11) and a matrix shape—defined by all indices within the dimensions—out of null values (lines 12-14). Then, an outer join (lines 11/15) combines both tables and assigns zero to missing values (`coalesce`: line 10).

```

1 create table if not exists iris (id serial, sepal_length float, sepal_width float, petal_length float,
  petal_width float, species int);
2 copy iris from './iris.csv' delimiter ',' HEADER CSV;
3 create table if not exists img (i int, j int, v float);
4 create table if not exists one_hot(i int, j int, v int);
5 insert into img (select id,1,sepal_length/10 from iris);
6 insert into img (select id,2,sepal_width/10 from iris);
7 insert into img (select id,3,petal_length/10 from iris);
8 insert into img (select id,4,petal_width/10 from iris);
9 insert into one_hot(
10   select n.i, n.j, coalesce(i.v,0), i.v
11   from (select id,species+1 as species,1 as v from iris) i right outer join
12     (select a.a as i, b.b as j from
13       (select generate_series as a from generate_series(1,select count(*) from iris)) a,
14       (select generate_series as b from generate_series(1,4)) b
15     ) n on n.i=i.id and n.j=i.species order by i,j);

```

List. 4: Data transformation: Feature matrix `img` and one-hot-encoded label `one_hot`.

After having transformed the data, we can create and initialise the weights again in relational representation. Using `generate_series` according to the matrix dimensions together with `random`, we initialise all required weights matrices.

```

1 create table if not exists w_xh (i int, j int, v float);
2 create table if not exists w_ho (i int, j int, v float);
3 insert into w_xh (select i.*,j.*,random()*2-1 from generate_series(1,4) i, generate_series(1,20) j);
4 insert into w_ho (select i.*,j.*,random()*2-1 from generate_series(1,20) i, generate_series(1,3) j);

```

List. 5: Create and initialise weights in SQL-92.

The feature matrix in relational representation forms the input for training the neural network within a recursive CTE (List. 6) that computes the weights per iteration of gradient

descent. As we need to compute all weights within the recursive CTE, a unique number (*id*) identifies each weight matrix. Thus a union of all weight matrices forms the base case for the recursion. Within the recursive step, nested CTEs help to evaluate the model (lines 6-15), to backpropagate the loss (lines 16-29) and to compute the derivative per weight matrix (lines 30-37). The first CTE *w\_now*—just referring to the original weights—is necessary, as PostgreSQL only allows one reference to the recursive table. Each following CTE computes one matrix operation, so either a matrix or a Hadamard multiplication, whose CTE name refers to the variable name (cf. Section 2.1). Finally, the weights were updated by subtracting their derivatives (lines 39-41).

```

1 with recursive w (iter,id,i,j,v) as (
2   (select 0,0,* from w_xh union select 0,1,* from w_ho)
3   union all
4   ( with w_now as ( -- recursive reference only allowed once in PSQL
5     select * from w
6   ), a_xh(i,j,v) as ( -- sig(img * w_xh)
7     select m.i, n.j, 1/(1+exp(-SUM (m.v*n.v)))
8     from img as m inner join w_now as n on m.j=n.i
9     where n.id=0 and n.iter=(select max(iter) from w_now) -- w_xh
10    group by m.i, n.j
11  ), a_ho(i,j,v) as ( -- sig(a_xh * w_ho)
12    select m.i, n.j, 1/(1+exp(-SUM (m.v*n.v)))
13    from a_xh as m inner join w_now as n on m.j=n.i
14    where n.id=1 and n.iter=(select max(iter) from w_now) -- w_ho
15    group by m.i, n.j
16  ), l_ho(i,j,v) as ( -- 2 * (a_ho-y_ones)
17    select m.i, m.j, 2*(m.v-n.v)
18    from a_ho as m inner join one_hot as n on m.i=n.i and m.j=n.j
19  ), d_ho(i,j,v) as ( -- l_ho * a_ho * (1-a_ho)
20    select m.i, m.j, m.v*n.v*(1-n.v)
21    from l_ho as m inner join a_ho as n on m.i=n.i and m.j=n.j
22  ), l_xh(i,j,v) as ( -- d_ho * w_ho^T
23    select m.i, n.i as j, SUM (m.v*n.v)
24    from d_ho as m inner join w_now as n on m.j=n.j
25    where n.id=1 and n.iter=(select max(iter) from w_now) -- w_ho
26    group by m.i, n.i
27  ), d_xh(i,j,v) as ( -- l_xh * a_xh * (1-a_ho)
28    select m.i, m.j, m.v*n.v*(1-n.v)
29    from l_xh as m inner join a_xh as n on m.i=n.i and m.j=n.j
30  ), d_w(id,i,j,v) as (
31    select 0, m.j as i, n.j, SUM (m.v*n.v)
32    from img as m inner join d_xh as n on m.i=n.i
33    group by m.j, n.j
34    union
35    select 1, m.j as i, n.j, SUM (m.v*n.v)
36    from a_xh as m inner join d_ho as n on m.i=n.i
37    group by m.j, n.j
38  )
39  select iter+1, w.id, w.i, w.j, w.v - 0.01 * d_w.v
40  from w_now as w, d_w
41  where iter < 20 and w.id=d_w.id and w.i=d_w.i and w.j=d_w.j
42 )
43 )
44 select * from w;

```

List. 6: Training a neural network in SQL-92.

In order to predict the accuracy of the trained weights, an SQL query measures the number of correctly classified labels (List. 7). Evaluating the model (lines 3-9) returns a vector of probabilities per tuple and category. The SQL query ranks the predicted probabilities per tuple (line 2) and the one-hot-encoded vector of the original labels (line 11) to compare whether the index of the highest probability matches the index of the one value (line 14). Although window functions were used for the ranking, they could be replaced by an anti-join using `not exists` to conform SQL-92.

```

1  select iter, count(*)::float/(select count(distinct i) from one_hot)
2  from ( select *, rank() over (partition by m.i,iter order by v desc)
3         from ( select m.i, n.j, 1/(1+exp(-sum (m.v*n.v))) as v, m.iter
4               from ( select m.i, n.j, 1/(1+exp(-sum (m.v*n.v))) as v, iter
5                     from img AS m inner join w as n on m.j=n.i
6                     where n.id=0
7                     group by m.i, n.j, iter ) AS m inner join w as n on m.j=n.i
8                     where n.id=1 and n.iter=m.iter
9                     group by m.i, n.j, m.iter
10              ) m ) pred,
11  (select *, rank() over (partition by m.i order by v desc) from one_hot m) test
12  where pred.i=test.i and pred.rank = 1 and test.rank=1
13  group by iter, pred.j=test.j
14  having (pred.j=test.j)=true
15  order by iter

```

List. 7: Prediction in SQL:2003 (with window functions).

### 3 Evaluation

*System:* Ubuntu 22.04 LTS, 20 Intel Xeon E5-2660 v2 CPU with hyper-threading, running at 2.20 GHz with 256 GB DDR4 RAM.

We compare the performance of the relational representation for matrices (*SQL-92*, List. 6) to their representation as an array data type [Sc21c] (*SQL + Arrays*). We apply both representations for use within neural networks in SQL and let the benchmarks<sup>4</sup> run in Umbra [NF20] and PostgreSQL (PSQL) 14.5 [SR86] as target engines. The implementation with NumPy (List. 1) serves as the baseline. We use two different data sets: Fisher’s Iris flower data [Fi36] (four attributes, one label) and the MNIST data [CMS12] for image classification (ten categories, 784 pixels).

#### 3.1 Scaling the Number of Input Tuples

Figure 4 shows the first benchmark on the Iris data set. As we are interested in the performance numbers and not in the model quality, we replicate the Iris flower data set for the first benchmark to enable a flexible input size. A neural network with one hidden

<sup>4</sup> <https://gitlab.db.in.tum.de/MaxEmanuel/nn2sql>

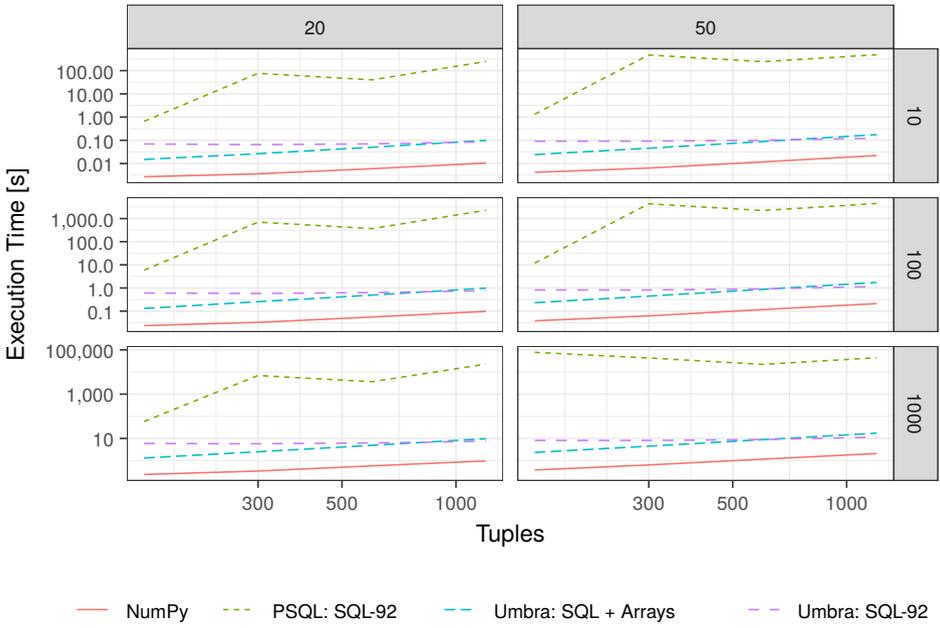


Fig. 4: Runtime for training a neural network with one hidden layer (size 20/50, 10/100/1000 iteration).

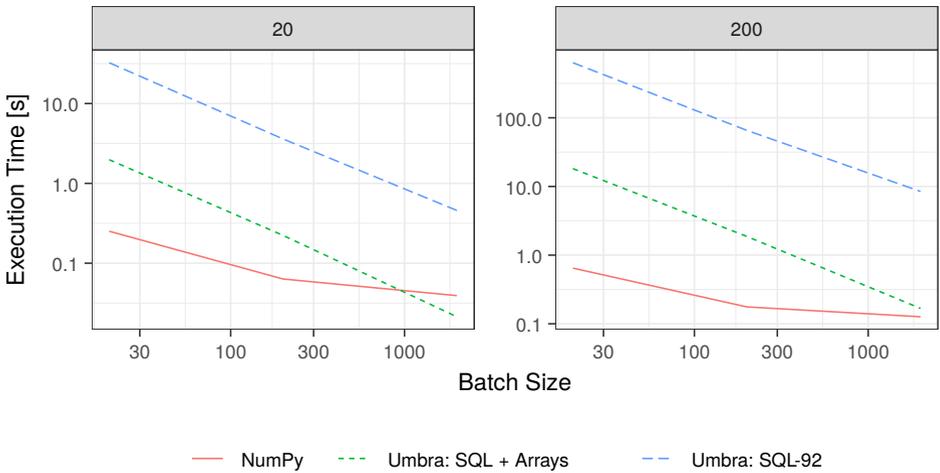


Fig. 5: Runtime for training one epoch with the MNIST data set with increasing batch size (one hidden layer size 20/200).

layer is trained to classify the flower category. We vary the size of the training data set, the number of iterations and the size of the hidden layer. Although the NumPy implementation outperforms both SQL variants, the performance increase of Umbra with its in-memory performance in comparison to PostgreSQL is visible. With only four attributes, the overhead for array operations dominates, so the relational representation performs better than the array data type for larger input data. Both SQL variants perform better with an increasing number of tuples per iteration. A small number of input tuples corresponds to a small batch size, leading to a small number of tuples used during one recursive step. This thwarts database systems as they excel in batched processing.

### 3.2 Image Classification

The second benchmark simulates image classification based on the MNIST data set using a neural network with one hidden layer. We measure the runtime for training one epoch depending on the batch size. As we can see in Figure 5, database systems perform better the bigger the batch size is. With a larger batch size, the runtime of the SQL implementations approximates the one of the baseline implementation. As the MNIST data set contains more attributes than the latter, the cost for aggregation into arrays is amortised and the SQL array data type outperforms the relational representation. To conclude, in-memory database systems are able to carry out matrix operations as required for neural networks. Nevertheless, use-case-specific optimisations are needed to support smaller batch sizes.

## 4 Conclusion

This paper has discussed and benchmarked building blocks for training neural networks in SQL. In order to deduce the necessary SQL queries that represent matrix algebra for evaluating and training neural networks, we first discussed reverse mode automatic differentiation to reuse partial derivations. The partial derivations formed the foundation for nested CTEs. They were cached within a recursive CTE when deriving the weight matrices to compute the optimal weights. In the evaluation, in-memory enhanced database systems, i.e. Umbra, showed comparable performance to state-of-the-art libraries used in machine learning, i.e. NumPy in Python, when training with larger batch sizes only.

Future research is required on optimising recursive CTEs for this use case and on automatically generating the presented queries. As we are using recursion to imitate a procedural loop, the recursive CTE grows with each iteration. Therefore, the memory consumption increases per iteration, which restricts the number of iterations and the model size. To overcome the restrictions, database optimisers should either detect subsequent selections to eliminate intermediate results within the CTE or output intermediate results to free memory. Assuming these optimisations, one can use the presented queries to train more complex models with more weight variables.

## References

- [Bl22] Blacher, M.; Giesen, J.; Laue, S.; Klaus, J.; Leis, V.: Machine Learning, Linear Algebra, and More: Is SQL All You Need? In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2022.
- [Bu20] Butterstein, D.; Martin, D.; Stolze, K.; Beier, F.; Zhong, J.; Wang, L.: Replication at the Speed of Change - a Fast, Scalable Replication Solution for Near Real-Time HTAP Processing. *Proc. VLDB Endow.* 13/12, pp. 3245–3257, 2020.
- [Bu22] Budach, L.; Feuerpfeil, M.; Ihde, N.; Nathansen, A.; Noack, N. S.; Patzlaff, H.; Harmouch, H.; Naumann, F.: The Effects of Data Quality on ML-Model Performance. *CoRR abs/2207.14529*, 2022.
- [CMS12] Ciresan, D. C.; Meier, U.; Schmidhuber, J.: Multi-column deep neural networks for image classification. In: *CVPR*. IEEE Computer Society, pp. 3642–3649, 2012.
- [DMG22] Dietrich, B.; Müller, T.; Grust, T.: Data provenance for recursive SQL queries. In: *TaPP*. *ACM*, 9:1–9:8, 2022.
- [Fi36] Fisher, R. A.: The use of multiple measurements in taxonomic problems. *Annals of eugenics* 7/2, pp. 179–188, 1936.
- [He22] Heinrich, R.; Luthra, M.; Kornmayer, H.; Binnig, C.: Zero-shot cost models for distributed stream processing. In: *DEBS*. *ACM*, pp. 85–90, 2022.
- [Ma15] May, N.; Lehner, W.; P., S. H.; Maheshwari, N.; Müller, C.; Chowdhuri, S.; Goel, A. K.: SAP HANA - From Relational OLAP Database to Big Data Infrastructure. In: *EDBT*. *OpenProceedings.org*, pp. 581–592, 2015.
- [MAF21] Miedema, D.; Aivaloglou, E.; Fletcher, G.: Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In: *ICER*. *ACM*, pp. 355–367, 2021.
- [MD22] Maltry, M.; Dittrich, J.: A Critical Analysis of Recursive Model Indexes. *Proc. VLDB Endow.* 15/5, pp. 1079–1091, 2022.
- [Mu17] Murray, I.: Machine Learning and Pattern Recognition (MLPR): Backpropagation of Derivatives, 2017, URL: [https://www.inf.ed.ac.uk/teaching/courses/mlpr/2017/notes/w5a\\_backprop.pdf](https://www.inf.ed.ac.uk/teaching/courses/mlpr/2017/notes/w5a_backprop.pdf), visited on: 09/02/2021.
- [Na22] Nath, R. P. D.; Romero, O.; Pedersen, T. B.; Hose, K.: High-level ETL for semantic data warehouses. *Semantic Web* 13/1, pp. 85–132, 2022.
- [NF20] Neumann, T.; Freitag, M. J.: Umbra: A Disk-Based System with In-Memory Performance. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2020.
- [OVZ22] Olteanu, D.; Vortmeier, N.; Zivanovic, D.: Givens QR Decomposition over Relational Databases. In: *SIGMOD Conference*. *ACM*, pp. 1948–1961, 2022.
- [Ra18] Raasveldt, M.; Holanda, P.; Mühleisen, H.; Manegold, S.: Deep Integration of Machine Learning Into Column Stores. In: *EDBT*. *OpenProceedings.org*, pp. 473–476, 2018.

- [Re22] Renz-Wieland, A.; Gemulla, R.; Kaoudi, Z.; Markl, V.: NuPS: A Parameter Server for Machine Learning with Non-Uniform Parameter Access. In: SIGMOD Conference. ACM, pp. 481–495, 2022.
- [Sa22] Salimzadeh, S.; Gadiraju, U.; Hauff, C.; van Deursen, A.: Exploring the Feasibility of Crowd-Powered Decomposition of Complex User Questions in Text-to-SQL Tasks. In: HT. ACM, pp. 154–165, 2022.
- [Sc19] Schüle, M. E.; Passing, L.; Kemper, A.; Neumann, T.: Ja-(zu-)SQL: Evaluation einer SQL-Skriptsprache für Hauptspeicherdatenbanksysteme. In: BTW. Vol. P-289. LNI, Gesellschaft für Informatik, Bonn, pp. 107–126, 2019.
- [Sc21a] Schuhknecht, F. M.; Priesterroth, A.; Henneberg, J.; Salkhordeh, R.: AnyOLAP: Analytical Processing of Arbitrary Data-Intensive Applications without ETL. Proc. VLDB Endow. 14/12, pp. 2823–2826, 2021.
- [Sc21b] Schüle, M. E.; Götz, T.; Kemper, A.; Neumann, T.: ArrayQL for Linear Algebra within Umbra. In: SSDBM. ACM, pp. 193–196, 2021.
- [Sc21c] Schüle, M. E.; Lang, H.; Springer, M.; Kemper, A.; Neumann, T.; Günne-  
mann, S.: In-Database Machine Learning with SQL on GPUs. In: SSDBM. ACM, pp. 25–36, 2021.
- [Sc21d] Schüle, M. E.; Schmeißer, J.; Blum, T.; Kemper, A.; Neumann, T.: TardisDB: Extending SQL to Support Versioning. In: SIGMOD Conference. ACM, pp. 2775–2778, 2021.
- [Sc22] Schüle, M. E.; Götz, T.; Kemper, A.; Neumann, T.: ArrayQL Integration into Code-Generating Database Systems. In: EDBT. OpenProceedings.org, 1:40–1:51, 2022.
- [SK22] Störl, U.; Klettke, M.: Darwin: A Data Platform for Schema Evolution Management and Data Migration. In: EDBT/ICDT Workshops. Vol. 3135. CEUR Workshop Proceedings, CEUR-WS.org, 2022.
- [SR86] Stonebraker, M.; Rowe, L. A.: The Design of Postgres. In: SIGMOD Conference. ACM Press, pp. 340–355, 1986.
- [WGR20] Wingerath, W.; Gessert, F.; Ritter, N.: InvalidDB: Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases (Extended). Proc. VLDB Endow. 13/12, pp. 3032–3045, 2020.
- [WH21] Wiese, L.; Höltje, D.: NNCompare: a framework for dataset selection, data augmentation and comparison of different neural networks for medical image analysis. In: DEEM@SIGMOD. ACM, 6:1–6:7, 2021.
- [WP22] Wenig, P.; Papenbrock, T.: DataGossip: A Data Exchange Extension for Distributed Machine Learning Algorithms. In: EDBT. OpenProceedings.org, 2:373–2:377, 2022.

# On the State of German (Abstractive) Text Summarization

Dennis Aumiller,<sup>1</sup> Jing Fan,<sup>2</sup> Michael Gertz<sup>3</sup>



**Abstract:** With recent advancements in the area of Natural Language Processing, the focus is slowly shifting from a purely English-centric view towards more language-specific solutions, including German. Especially practical for businesses to analyze their growing amount of textual data are text summarization systems, which transform long input documents into compressed and more digestible summary texts. In this work, we assess the particular landscape of German abstractive text summarization and investigate the reasons why practically useful solutions for abstractive text summarization are still absent in industry.

Our focus is two-fold, analyzing a) training resources, and b) publicly available summarization systems. We are able to show that popular existing datasets exhibit crucial flaws in their assumptions about the original sources, which frequently leads to detrimental effects on system generalization and evaluation biases. We confirm that for the most popular training dataset, MLSUM, over 50% of the training set is unsuitable for abstractive summarization purposes. Furthermore, available systems frequently fail to compare to simple baselines, and ignore more effective and efficient extractive summarization approaches. We attribute poor evaluation quality to a variety of different factors, which are investigated in more detail in this work: A lack of qualitative (and diverse) gold data considered for training, understudied (and untreated) positional biases in some of the existing datasets, and the lack of easily accessible and streamlined pre-processing strategies or analysis tools. We therefore provide a comprehensive assessment of available models on the cleaned versions of datasets, and find that this can lead to a reduction of more than 20 ROUGE-1 points during evaluation. As a cautious reminder for future work, we also highlight the problems of solely relying on  $n$ -gram based scoring methods by presenting particularly problematic failure cases. The code for dataset filtering and reproducing results can be found online: <https://github.com/dennlinger/summaries>

**Keywords:** Abstractive Text Summarization; Natural Language Generation; German; Evaluation

## 1 Introduction

Libraries simplifying the access to pre-trained neural models have greatly pushed the recent advancement of state-of-the-art performance in many tasks [Wo20]. However, with the general absence of non-English resources, one of the prevalent challenges in the Natural Language Processing (NLP) community is the extension of approaches to other languages beyond English. Subsequently, evaluation quality and consistency is even harder to maintain

<sup>1</sup> Heidelberg University, Institute of Computer Science, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany  
aumiller@informatik.uni-heidelberg.de

<sup>2</sup> Heidelberg University, Institute of Computer Science, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany  
j.fan@stud.uni-heidelberg.de

<sup>3</sup> Heidelberg University, Institute of Computer Science, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany  
gertz@informatik.uni-heidelberg.de

in setups, where high-quality gold data is scarce. This can lead to unintended consequences during the interpretation of model performance and generalization capabilities beyond narrow domain-specific use cases.

A sub-task of the NLP community that deserves particular attention is text summarization. The focus here is to produce an abridged version of an input text that accurately *summarizes* the key points of the original text. Such systems offer an immediate benefit in times with ever-increasing amounts of textual information, and allow users to quickly grasp the contents of even complex documents. In particular, we differentiate between various sub-tasks of text summarization: *extractive* systems provide summaries by simply copying text snippets from the original input, which is efficient to compute, but comes at the cost of lower textual fluency. On the other hand, *abstractive* summarization systems may introduce new phrases, or even full sentences, which are not present in the original document. This potentially increases a summary’s fluency and conciseness over extractive methods. Abstractive text summarization systems are generally built upon the more recent development of sequence-to-sequence neural models [SVL14, BCB15], which come with an exploding computational cost.

Particularly for (abstractive) summarization, the previously mentioned issues of data scarcity for non-English methods are further worsened by a lack of diverse (and readily available) evaluation metrics. Most works rely entirely on  $n$ -gram-based analysis of system summaries, such as ROUGE [Li04], which cannot accurately judge the truthfulness of a generated summary, i.e., how accurately the original text’s factual statements are represented in the generated summary. Only few works extend their evaluation to human result inspections, given its higher cost. However, there are several critical assumptions that –even under basic premises– are exposed as oftentimes insufficient for a comprehensive analysis [SJ97, tHKdR20]. Examples are the focus on singular target summaries, ignoring the subjective nature of differing viewpoints of annotators, as well as the focus on particularly prominent sentences in the first few paragraphs of reference articles [Zh21b].

In this work, we focus on German abstractive summarization systems and set out to investigate, reproduce, and evaluate summarization systems. In conjunction to a model-centric view of summarization, we further review the existing training resources for German, including their particular domain and data curation processes.

1. We find that in particular automatically created and multilingual resources suffer from insufficient pre-processing, potentially due to the absence of a native speaker during the curation process.
2. News documents seem to be overly represented in trained systems, potentially due to a popularity bias in English summarization datasets for news resources.
3. Baseline scores are heavily affected by data biases in test sets of prominent datasets.

Upon conducting a qualitative analysis of outputs from publicly available models, we further find that most systems fall severely short of the expected quality in at least one of the following areas:

1. Due to positional biases, text snippets may be directly copied from the beginning of the input text, constituting an *extractive* instead of an *abstractive* summary. Especially considering the computational requirements of neural systems being orders of magnitudes greater than simple extractive summarizers, this undermines the quality of neural text generations.
2. Generated outputs may contain (severe) syntactic errors, to the point of becoming illegible or hard to interpret.
3. Semantic mistakes introduce factual errors, leading to incorrect conclusions from the summary alone. This problem is exacerbated for longer input documents, where a structured content understanding is necessary to maintain factual consistency.

For data-centric issues, current pipelines are not taking user-specified filtering steps into account; oftentimes, datasets are directly used “out of the box”, without any further data verification step involved. For this purpose, we extend available summarization-specific filtering steps and provide a simple-to-use and language-agnostic processing library.

For model-centric problems, it is near impossible to identify failure cases with existing metrics; costly manual inspection of individual samples would be required. Simultaneously, we work towards expanding the available scores to help facilitate a better understanding of current expectations towards summarization systems. In the following, we will briefly mention work on automated evaluation of summarization systems, including a comprehensive look at the current landscape of German abstractive summarization; we follow with a formal introduction of our proposed filtering methods for summarization datasets, as well as a list of model-centric checks to consider. We discuss exhibited quality issues in existing datasets and systems for German summarization, and conclude with a brief outlook for future work.

## 2 Related Work

We establish an extensive overview of currently available training resources for German summarization systems and survey the landscape of trained models, with a particular focus on publicly available methods.

Aside from this, we further reiterate some of the common pitfalls in evaluating summarization systems, which will become particularly relevant during the experiments in this work.

## 2.1 German Data Sources for Summarization

In our experiments, we focus on seven different datasets across a variety of domains. To our knowledge, these cover all of the publicly available sources used for training German systems.

**MLSUM [Sc20]** This multilingual dataset was presented as one of the first efforts in making larger-scale training sets available for multiple languages that also include German as a language. MLSUM is constructed by extracting news articles and associated summary sections as generation targets. We use the German subset in this work, which is by far the most popular dataset used for training and evaluating resources in German, based on our survey. Despite its popularity, issues in the quality of samples have gone unnoticed until early 2022, when Philip May [Ma22] was the first to report on problems with fully extractive summaries, an aspect we will analyze in more detail later.

**MassiveSumm [VS21]** The construction of this particular dataset is similar to MLSUM and focuses on a large number of automatically extracted summaries from news articles in multiple languages. The authors perform some rudimentary filtering with respect to empty samples and even go as far as avoiding similar issues to MLSUM by removing what they call “ellipsoid summaries”, i.e., fully extractive summaries that appear at the beginning of the reference text. While the quality of the samples is comparatively low due to the automated extraction process, this corpus is by far the largest considered, with around 480,000 samples, and has the potential to improve existing training setups with its sheer number of samples.

**Swisstext [FVM20, Fr20]** In contrast to the –generally shorter– news articles available in MLSUM, the Swisstext dataset provides longer-form summaries based on German Wikipedia pages, which has been later extended to the GeWiki corpus [Fr20]. For the construction, the central argument is that the introductory paragraph serves as a “summary” of the remaining article text. The provided dataset comes with a training portion and a private test set, meaning no ground truth summaries are available for the test samples. A multilingual variant of this idea, the XWikis corpus, was introduced shortly after [PBL21]. While the XWikis corpus contains more samples per language, including for German, monolingual data is not readily available for download. Adding the fact that German summarization works primarily deal with the Swisstext dataset, we choose the latter for our experiments.

**Klexikon [AG22]** Another Wikipedia-related resource, but with different target summaries. Instead of utilizing a page’s introductory paragraph, the authors align articles from a simplified children’s encyclopedia (Klexikon) on the same topic. Consequently, this dataset has much longer summary lengths but covers a much smaller subset of only around 3,000

samples. Given the secondary focus on simplification in the target summaries, this corpus requires a considerably higher level of abstractive reformulations during the generation.

**WikiLingua [La20]** As the third multilingual resource, summaries in this corpus are extracted from the WikiHow platform. Here, Ladhak et al. [La20] consider short instruction summaries of individual steps in WikiHow guides and align those with the referenced paragraphs. The general tone of the dataset is rather informal and is in a more imperative style in comparison to other data sources. To align non-English samples, associated images are used to identify paragraphs occurring in different languages. Importantly, this means that for German articles, frequently only some of the article’s paragraphs are actually contained in the dataset.

**LegalSum [GMM21]** Another area benefiting enormously from high-quality summaries is the legal domain. LegalSum is the first German resource providing summaries of around 100,000 court rulings. On average, these samples require the highest amount of compression across evaluated datasets.

**EUR-Lex-Sum [ACG22]** As a secondary resource for legal texts, Aumiller et al. present a multilingual corpus based on EU-level legal acts, semi-aligned across languages. The corpus is considerably smaller than LegalSum, with only about 1,900 German documents available. While the EUR-Lex-Sum corpus has extremely long documents, it also presents a more challenging summary generation with the longest average summary length across all considered corpora (generally between 600-800 words). Importantly, summaries are also written by human expert annotators and therefore present a much higher-quality standard for summaries compared to some of the other datasets.

**Further Resources** In addition to these datasets, we are aware of at least two more news-related resources. One is used in experiments by Nitsche [Ni19], where data was supplied by the German Press Agency, but no public record of it exists. The second corpus is hinted at online by users on Huggingface’s platform<sup>4</sup>.

For clinical summarization, Liang et al. [Li22] present a resource of about 11,000 radiology reports; given the sensitive nature of the data, no publicly available version exists as of now. We are also aware of a secondary source of the WikiLingua dataset by GEM<sup>5</sup>, which provides additional samples, as well as a pre-split validation and test section not provided in the original German subset. In preliminary experiments, we found that > 99.89% of the data were valid samples for the GEM source. Most problematic is the automatic combination of

<sup>4</sup> A news corpus with ca. 400,000 articles is indicated here: [https://huggingface.co/Einmalumdiewelt/PegasusXSUM\\_GNAD/discussions/1#6308eb5037556c4ab03258df](https://huggingface.co/Einmalumdiewelt/PegasusXSUM_GNAD/discussions/1#6308eb5037556c4ab03258df), last accessed: 2023-01-14

<sup>5</sup> [https://gem-benchmark.com/data\\_cards/wiki\\_lingua](https://gem-benchmark.com/data_cards/wiki_lingua), last accessed: 2023-01-14

paragraphs into one summary, which can cause disjoint reference texts or summaries. Finally, all of the discussed corpora so far are types of single document summarization resources, where a summary is extracted from a *singular text only*. Datasets for training summarization systems that consider multiple source texts exist at smaller scales, but require further manual adjustment for acquisition [Be16, Zo18]. More recent experiments with neural models utilizing the latter corpus have been conducted by Johner et al. [JJB21].

## 2.2 German Summarization Systems

| Model                            | Training data                               | Test Set  | Evaluation        | Filtering           | Public | Reprod. |
|----------------------------------|---|-----------|-------------------|---------------------|--------|---------|
| mrm8488/bert2bert <sup>6</sup>   | MLSUM                                       | MLSUM     | ROUGE             | None                | ✓      | ✓       |
| ml6team/mt5-small <sup>7</sup>   | MLSUM                                       | MLSUM     | ROUGE             | Length              | ✓      | ✗       |
| T-Systems/mt5-small <sup>8</sup> | CNN/DailyMail,<br>MLSUM, XSum,<br>Swisstext | MLSUM     | ROUGE             | Length &<br>Overlap | ✓      | ✗       |
| Shahm/t5-small <sup>9</sup>      | MLSUM                                       | MLSUM     | ROUGE             | None                | ✓      | ✗       |
| T5-base <sup>10</sup>            | ?   | ?         | ROUGE             | ?                   | ✓      | ✗       |
| german-t5 <sup>11</sup>          | Swisstext                                   | MLSUM     | ROUGE             | ?                   | ✗      | ✗       |
| BERT-Copy [Ak20]                 | Swisstext                                   | Swisstext | ROUGE &<br>manual | ?                   | ✓      | ?       |
| Transformer [PM19]               | Swisstext &<br>CommonCrawl                  | Swisstext | ROUGE &<br>manual | None                | ✗      | ✗       |
| Fact-Encoder [Ve19]              | Swisstext                                   | Swisstext | ROUGE &<br>manual | None                | ✗      | ✗       |
| Pointer-Gen [FBZ19]              | Swisstext                                   | Swisstext | ROUGE &<br>manual | ?                   | ✗      | ✗       |
| Enc-Dec [GMM21]                  | LegalSum                                    | LegalSum  | ROUGE             | ?                   | ✓      | ?       |
| bert2bert [Li22]                 | Radiology                                   | Radiology | ROUGE &<br>manual | ?                   | ✗      | ✗       |

Tab. 1: List of German abstractive summarization systems. We detail their known properties from training recipes or papers. If we have access to models, we denote whether public scores are reproducible within  $\pm 0.5$  ROUGE points (“*Reprod.*”); ? in the reproducibility column indicates models that are available, however, we were unable to successfully run locally.

While we are slowly starting to see a greater diversity in the available training resources for German text summarization, it comes as a small surprise that the availability of trained system is much less diverse. As will become more apparent in later sections, the primary focus for training systems is a combination of a pre-trained checkpoint and one predominant training resource (“MLSUM”, particularly the German subset). Below, we elaborate on

considered model properties, differentiating between the availability levels of related works and their backgrounds. A summary of known properties can be seen in Table 1.

### 2.2.1 Publicly Available Systems

The primary source for available models is the Huggingface Hub<sup>12</sup>, which allows filtering by supported language and appropriate task (in our case summarization). We note that some of the available models are not properly tagged, but spent considerable time to ensure no other models were accidentally ignored. For users who have uploaded several different versions, we selected the model with the highest self-reported evaluation scores.

Given that users on the platform are likely familiar with other services of Huggingface (including their datasets browser), it comes as no surprise that the diversity between models and training setups is low. The primary choice falls on either mT5 [Xu21] or variants of T5 [Ra20], with some alternatives based on (m)BART [Le20, Li20] being consistently outperformed according to self-reported metrics by authors. In order to train effectively on large quantities on data, most approaches use one of the smaller checkpoints, referring to model variants with fewer parameters. Outside of the model hub, code repositories exist for the BERT-Copy architecture by Aksenov et al. [Ak20] and Encoder-Decoder models used by Glaser et al. [GMM21]. However, we were unable to set up inference for custom datasets based on the respective code bases.

### 2.2.2 Private Models

A further selection of models has been published in response to the Swisstext 2019 summarization challenge [PM19, Ve19, FBZ19]. However, neither team has published any associated public repository. Similarly, no models are available from Liang et al.’s work on radiology reports [Li22]. As the only one of the major cloud providers, Microsoft offers an extractive summarization service through Azure that supports German.<sup>13</sup> Otherwise, the only commercial solution providing a platform for abstractive summarization also supporting German texts is currently Aleph Alpha.<sup>14</sup>

<sup>6</sup> [https://hf.co/mrm8488/bert2bert\\_shared-german-finetuned-summarization](https://hf.co/mrm8488/bert2bert_shared-german-finetuned-summarization), last accessed: 2022-10-06

<sup>7</sup> <https://huggingface.co/ml6team/mt5-small-german-finetune-mlsum>, last accessed: 2022-10-06

<sup>8</sup> <https://huggingface.co/T-Systems-onsite/mt5-small-sum-de-en-v2>, last accessed: 2022-10-06

<sup>9</sup> <https://huggingface.co/Shahm/t5-small-german>, last accessed: 2022-10-06

<sup>10</sup> [https://huggingface.co/Einmalumdiewelt/T5-Base\\_GNAD](https://huggingface.co/Einmalumdiewelt/T5-Base_GNAD), last accessed: 2022-10-06

<sup>11</sup> <https://github.com/GermanT5/german-t5-eval>, last accessed: 2022-10-06

<sup>12</sup> <https://huggingface.co/models>, last accessed: 2023-01-14

<sup>13</sup> <https://learn.microsoft.com/en-us/azure/cognitive-services/language-service/summarization/language-support>, last accessed: 2022-10-06

<sup>14</sup> <https://www.aleph-alpha.com/use-cases/conversion#trilingual-summary>, last accessed: 2022-10-06

### 2.3 Evaluation Metrics for Summarization

As previously mentioned, the de-facto gold standard for evaluating summarization systems is the usage of ROUGE [Li04]. The authors introduce unigram overlap (ROUGE-1), bigram overlap (ROUGE-2) and the longest common subsequence (ROUGE-L) between system and gold predictions. The underlying core assumption is based on  $n$ -gram co-occurrences in the generated text with respect to one or more gold summaries. The fact that ROUGE can handle several reference samples at the same time is crucial for understanding some of the implications in the later parts of this work: with several references, variation in wording, e.g., particular expressions, are much easier to compare against than in a single reference summary. However, despite the theoretical support for multi-labels, few datasets ever provide such costly annotations.

In turn, more recently proposed alternatives to ROUGE rely on score computation from a single gold summary only [ECM19]. Examples include primarily neural similarity scoring between a generated summary and a gold reference [SDP20, Zh20]. Ultimately, neural methods are also incredibly expensive to employ for evaluation settings, potentially taking several days to evaluate a single test set [Na21]. Besides the cost factor, the main issue with such alternative scores is two-fold: On the one hand, a distinct advantage of simple co-occurrence-based metrics such as ROUGE is the simplicity in transferring it to another language. Even basic extensions, such as stemmers, are readily available in a multitude of languages other than English. Trained metrics, such as BERTScore [Zh20] or QAEval [DBWR21], however, are severely limited in their transferability to other languages, and would require dedicated efforts to port them to German, for example. On the other hand, recent statistical analyses have shown that when accounting for annotator expertise, correlation can vary significantly [Fa21]. When additionally controlling for variance and confidence intervals, correlation with human judgments over ROUGE correlation is only statistically significant in rare cases [DDR21]. A particular investigation on metrics for German summarization was conducted during the second Swisstext challenge [FVM20]. However, submitted resources were only marginally better than ROUGE baselines for judging system quality [PC20, Bi20]. For crowd-sourced evaluation approaches, Iskender et al. [IPM20] further elaborate on the importance of survey setups and considerations for expert annotators to ground evaluation results.

## 3 Assessing the Quality of Summarization Systems

When using existing models for abstractive text summarization, the expectation is that they should work “as expected”, meaning that a model provides appropriate and correct summaries. However, in practice, the automated collection of samples may lead to insufficient sample quality or systematic biases in the data. This has further detrimental consequences for models trained on those datasets.

In this section, we lay out a series of very basic sanity checks for both data and models, which help to ensure a minimal level of generalization from experimental results. As we will

| Issue                  | Reference  | Summary  |
|------------------------|--|--|
| <b>Short text</b>      | Wir verwenden Cookies, um unser Angebot für Sie zu verbessern. Mehr Informationen [...]. | –  |
| <b>Duplicates</b>      | ‘Virtuelles Bergsteigen mit dem Project360 [...]   | Leben und Kultur in Europa                     |
|                        | Historische Dokumente: Bilder der Wende [...]  | Leben und Kultur in Europa                     |
| <b>Relative Length</b> | Chef-Sprüche: “Ich sehe meine Kinder auch nur im Urlaub.”                                | Die besten Chef-Sprüche zum Thema Überstunden. |

Fig. 1: List of faulty summarization samples in the MassiveSumm dataset uncovered by various data checks. Despite checking for unrelated issues, we notice a trend where filtered samples are of especially low semantic quality, too.

later find, even such basic data assurances lead to a significant reduction of valid samples in available German summarization data.

### 3.1 Data-centric Sanity Checks

The best strategy to achieve decent experimental results is ensuring high quality in the training data – in line with the popular saying “garbage in, garbage out”. We present a list of minimal quality checks for individual samples, as well as dataset-wide assurances of data quality. Most of these measures are fully automated and at most require single hyperparameter settings to filter datasets.

Further, suggested data checks are language-independent at their core and can therefore be applied in basic form to any dataset, even beyond German. This also implies that no further existing tools or libraries for tokenization, etc., are required.

**Empty Samples** The most trivial sanity check is verifying that *both* the reference text and summary are present for all samples. This is simultaneously the most prevalent check implemented by authors of resource papers in our experience. Even so, several issues can arise for this criterion, primarily revolving around varying definitions of “emptiness”. For example, one could also consider a sample as empty if only whitespaces (or whitespace-like symbols, such as `\t`) are present. Extensions are, for example, faulty encodings or only special characters in a text (cf., data audit insights by Kreutzer et al. [Kr22]).

**Minimum Text Length** A superset of “empty samples”, imposing a required minimum text length presents a stricter filtering criterion for sample validity. Where empty texts are universally to be avoided, hard length requirements are harder to impose, since the

appropriate cutoff depends strongly on the dataset domain. For domain-specific datasets, e.g., the instruction-like texts in the WikiLingua dataset [La20], having extremely short summaries with only a few characters (and comparatively short references) may make sense. For summaries stemming from news articles, however, length requirements imposed on the reference might ensure a longer minimum text length for quality control.

**Compression Ratio Filtering** Another key metric used in summarization research is the *Compression Ratio (CR)*, defined as the relation between reference text length and summary length. We follow the definition by Grusky et al. [GNA18]:  $CR(\text{ref}, \text{summ}) = \frac{\text{len}(\text{ref})}{\text{len}(\text{summ})}$ . For filtering by compression ratio, a significant difference should be ensured by establishing a minimum compression ratio. For our purposes, we argue that a reduction of at least 20% in the summary length is required, which equals  $CR \geq 1.25$ . We note that this is not a strict requirement per se and may depend on domain-specific factors. It can be argued, however, that samples with summaries longer (or equal) than their respective references (i.e.,  $CR \leq 1.0$ ) always pose an inadequate sample and must be filtered.

Related work sometimes takes a more drastic approach to compression ratio filtering, arguing that extreme content reduction may result in a lossy summary and should therefore also be avoided [Ur22].

**Duplicate Removal** Some lesser-checked property seems to be the existence of duplicates in training data, which is also applicable in more general machine learning settings. However, given that each sample for summarization comes with *two* distinct texts (the reference and the summary), we can further distinguish between different instances of duplication. Trivial to consider are instances of what we call **exact duplicates**, i.e., samples that have the exact same combination of reference and summary appearing as another tuple in the dataset.

We can further expand this idea by three more considerations, which we call **partial duplicates**. These are instances where we find either the reference or summary in other dataset instances. Finally, it could also occur that both summary and reference are duplicated, but across different samples; such instances are also considered partial duplicates and are relatively rare.

To understand why duplicates, including partial ones, can be considered harmful as a training resource, we need only look at the potential effects during training or evaluation. For exact duplicates, no real gain is achieved by including one sample several times in the training data. Worse yet, if we encounter exact duplicates *across different splits*, this can cause active falsification of evaluation results (train-test leakage). While partial duplicates are less severe, we still encourage removal, as they can cause confusion during the learning process: cases where different input texts should generate the same summary hamper generalization of models, and the reverse case of similar input texts generating different summaries conveys unclear learning signals during training. Finally, we also want to note that partial duplicates can uncover incorrectly aligned samples (cf. Fig. 1).

While *spotting* duplicates is fairly straightforward, removing duplicate content is often non-trivial, as there exist several valid strategies for deduplication, leading to differing results. In an attempt to reduce impact on smaller test and validation sets, we adopt a “additive” strategy for the remainder of this work. We start with an empty dataset, and iteratively add new samples if and only if neither the reference nor the summary have been previously included.

**Sample Inspection** Even with all of the proposed automated measures, nothing can ensure data quality quite as well as manually inspecting data. All of the previous measures can point to systemic failures in the data collection process, but may ignore more localized quality issues for particular samples. While a manual analysis step is not feasible at scale, often enough reviewing few samples will already reveal tendencies about the underlying data quality. We generally differentiate between the following strategies to inspect data samples and their respective up- and downsides:

1. **Reviewing samples in order:** A linear sequence of samples may reveal particular issues in the consistency of samples, which can be linked to the crawling process. We emphasize that “linearity” can follow many particular axes, not just the order in which data is stored. Further possible orderings can be based on available metadata descriptors, such as sortings by timestamps, source or length. In-order samples are most likely to uncover systematic issues, such as incorrect alignment settings that span several samples.
2. **Reviewing random samples:** Another popular approach is to shuffle data and randomly select instances for review. This is fairly easy to implement and does not require iterating over the full dataset or sorting operations. Advantages of random reviews are a more holistic coverage of the data distribution, but requires potentially more manual reviews to find systemic failures.
3. **Outliers and representative samples:** If data statistics are already known or easy to compute, a more targeted approach is to look for distributional outliers. There are again a variety of metrics that can be considered, with the most obvious being text length and compression ratio of individual samples. Manually reviewing outliers can also sharpen the requirements of expected outputs, e.g., the minimum/maximum length of a summary in relation to the input text. Related are *representative samples*, which constitute instances close to the mean or median of a distribution.

### 3.2 Model-centric Checks

While we have compiled a detailed list of what can be done about checking the data used for summarization systems, it is significantly harder to judge a trained system, especially

given that many neural methods can only be treated as black box systems. But even with a lack of clarity around the original training procedures and model learnings, we can use several probing techniques to estimate the robustness and performance of systems.

**Evaluation on Cleaned Test Sets** The standard procedure to evaluate on withheld (but still in-domain) test sets. While these evaluation approaches may give insights on the overfitting of trained models, such experiments tend to fall short of giving more concrete evidence on the pattern of how summaries are generated. This is especially crucial if no further manual evaluation is performed. Testing models on modified or generalized test data can serve as a partial remedy to this, by probing the generalization ability of particular systems. In combination with the proposed filtering techniques, we suggest the evaluation on cleaned test sets for models that were trained on the unfiltered training set. The main advantage is that no additional re-training with altered training sets is required, and insights can be acquired from a generally much smaller evaluation set through inference alone. Further, we hypothesize that intrinsic summarization metrics [NCL18, GNA18, Zh19, BC20] applied to *system summaries* can be used as a preliminary gauge for text quality in comparison to the original input. Especially *abtractiveness* of generated outputs, essentially constituting the number of novel  $n$ -grams in summaries, could indicate changes in the vocabulary.

**Domain Generalization** An extreme case of the previous point is testing on completely out-of-domain data, which usually means taking test splits of a different dataset. While this approach can be useful to evaluate general purpose summarization systems, the evaluated models in this work all present rather focused domain-specific summarization systems. For this reason, we refrain from evaluating performance based on out-of-domain abilities.

**Factual Consistency** A rather important argument for summarization especially: facts that are stated in the original reference text must be maintained in the respective summary. Therefore, models should be measured with respect to their truthfulness, which has been previously attempted with automatic metrics for English summarization systems [Kr20], or even implemented as an optimization target for more truthful summaries [Zh21a].

### 3.3 Extractive Models and Baseline Systems

Given the relatively one-dimensional approach to evaluation, we should at least expect additional context for better interpretability of model scores. In practice though, we rarely find a consistent reporting of baseline scores, if any comparisons are reported at all. To this end, we strive to provide consistent baselines and reporting of such in the context of German abstractive summarization. In addition to the scores, baselines also serve an important purpose by providing a sensible complexity trade-off: Unlike most neural methods, they

should be able to generate summaries faster and with fewer parameters than heavy-weight state-of-the-art approaches. Similar to English works, we therefore fall back on extractive summarization systems, which – as the name indicates – simply copy text snippets from the reference to generate a summary.

To our knowledge, the only work that has explicitly worked on extractive summarization for German is over 20 years old [Re00]. This does not imply, however, that there is no dedicated extractive system available. Especially for unsupervised methods, such as TextRank [MT04] or LexRank [ER04], language-specific taggers or lemmatizers can easily be replaced in existing libraries to enable application on German texts as well. For our experiments, we rely on three variants of baselines, which extract a specified number of sentences from the input text to generate a summary. Overall, extractive summaries are guaranteed to ensure a more factually consistent summary, and have high intra-sentence coherence. On the other hand, these methods cannot be fine-tuned and rely on singular hyperparameters – the length of the generated summary. This can still significantly impact the evaluation performance, but does not factor in domain-specific variance in text distribution. For all systems, we rely on the sentence splitting module by spaCy<sup>15</sup>, unless datasets provide a pre-split sentence format.

**Lead-3** The simplest possible baseline system is *lead-3*, popularized by Nallapati et al. [NZZ17] as a simple but strong baseline for news article summarization. Here, the summary is equal to the first three sentences of an input text. The method works particularly well for news texts, where key information has to be conveyed early on to both inform and catch the interest of a potential reader. The prevalence of this so-called “lead bias” differs significantly across different domains.

**Lead- $k$**  For other domains, three sentences may underestimate the actual summary length. For this purpose, Aumiller and Gertz [AG22] introduce a variant that extends the lead baseline to the  $k$  leading sentences, in their particular context the full first paragraph of a Wikipedia page. Given that in general, datasets do not contain paragraph-level information, the authors later extended this baseline and instead consider an approximate  $\hat{k}$  for each sample by using the average compression ratio [ACG22]:

$$\hat{k} = \frac{\text{len}(\text{reference})}{CR_{\text{avg}}}, \quad (1)$$

where  $\text{len}(\text{reference})$  is the number of sentences in the summary, and  $CR_{\text{avg}}$  denotes the average compression ratio across the *training* split of a dataset.

**Modified LexRank (LexRank-ST)** A more complex baseline that also considers sentences at other positions of the article is a modification of LexRank [ER04], similarly used by

<sup>15</sup> We use the model `de_core_news_sm` in our experiments.

Aumiller et al. [AG22, ACG22]. The key modification lies in exchanging the centrality computation – which is originally based on pure occurrence counts – with dense sentence embeddings obtained through sentence-transformers [RG19, RG20]. While the underlying neural model can be of arbitrary complexity, it does not need to be trained further to work in the summarization application. After scoring individual sentences, the highest-ranking  $k$  sentences are selected as the summary; we use the same method for estimating an optimal length  $\hat{k}$  as for the lead- $k$  baseline.

Finally, we also point towards oracle extractive summaries as a form of upper-bound for extractive summarization, which can be computed from greedy ROUGE-2 alignments [NZZ17, GMM21]. Given that we focus on *abstractive* results in this work, we omit the computation of extractive oracle summaries.

## 4 A Sober Look at State-of-the-Art Results

Given the presented set of tools, we now set out to put current models’ capabilities into a better context. To this end, we conduct a set of four experiments: We start by applying the filters introduced in Sect. 3.1 to available German summarization datasets, noting varying size reductions as a result. To remedy the changes introduced by our filtering, we re-compute a set of strong baselines as updated results for datasets with available validation and test sets. Further, given the previously uncovered discrepancies in some datasets, we repeat more comprehensive experiments on MLSUM and MassiveSumm across the pre- and post-filtered dataset to highlight the effect of filtering on ROUGE scores. We are able to show that this change in data quality also significantly impacts the reproducibility of results. Finally, we provide a small case study in which we examine a subset of generated samples that highlight some of the particular model-centric issues.

### 4.1 Filtering Datasets

**Key Finding 1:** German subsets of two popular multilingual resources (MLSUM and MassiveSumm) have extreme data quality issues, affecting **more than 25% of samples** across all splits.

Table 2 presents our findings for filtering the available German summarization datasets; hyperparameters for filters are specified in the table caption. We refrain from imposing any particularly strict filtering metrics, particularly for the length of texts. Most concerning is the fraction of affected samples in MLSUM, given its popularity as a training resource for many public models. While a strong lead bias is to be expected due to the domain of these samples being exclusively news articles, the eventual performance of models trained on the unfiltered dataset is severely impacted; a finding that we confirm in subsequent experiments. Primarily, it indicates that for fully extractive samples, summaries can be generated by

| Dataset   | Split | Samples | Min Length |               | Id | Min CR        | Fully Extr     | Duplicates |               |              | Valid Samples |           |
|-----------|-------|---------|------------|---------------|----|---------------|----------------|------------|---------------|--------------|---------------|-----------|
|           |       |         | Ref        | Summ          |    |               |                | Exact      | Ref           | Summ         |               |           |
| MLSUM     | Train | 220,887 | 0          | 0             | 39 | 30            | <b>126,204</b> | 31         | 45            | 105          | 94,433        | (42.75%)  |
|           | Val   | 11,394  | 0          | 0             | 0  | 0             | <b>3,285</b>   | 1          | 1             | 5            | 8,102         | (71.11%)  |
|           | Test  | 10,701  | 0          | 0             | 0  | 0             | <b>3,306</b>   | 1          | 5             | 2            | 7,387         | (69.03%)  |
| MassiveS  | Train | 478,143 | 253        | <b>16,294</b> | 0  | <b>33,959</b> | 0              | 805        | <b>73,886</b> | 4,882        | 348,064       | (72.79%)  |
| Swisstext | Train | 100,000 | 0          | 0             | 0  | 0             | 3              | 0          | 0             | 2            | 99,995        | (100.00%) |
| WikiLing  | Train | 58,341  | 11         | 0             | 0  | <b>1,435</b>  | 0              | 4          | 2             | 52           | 56,837        | (97.42%)  |
| Klexikon  | Train | 2,346   | 0          | 0             | 0  | 10            | 0              | 0          | 2             | 0            | 2,334         | (99.49%)  |
|           | Val   | 273     | 0          | 0             | 0  | 1             | 0              | 0          | 0             | 0            | 272           | (99.63%)  |
|           | Test  | 274     | 0          | 0             | 0  | 1             | 0              | 0          | 0             | 0            | 273           | (99.64%)  |
| EUR-Lex   | Train | 1,115   | 0          | 0             | 0  | 18            | 0              | 0          | 0             | 0            | 1,097         | (98.39%)  |
|           | Val   | 187     | 0          | 0             | 0  | 0             | 0              | 0          | 0             | 0            | 187           | (100.00%) |
|           | Test  | 188     | 0          | 0             | 0  | 0             | 0              | 0          | 0             | 0            | 188           | (100.00%) |
| LegalSum  | Train | 79,937  | 0          | 2             | 0  | 12            | 326            | 233        | 95            | <b>3,106</b> | 76,163        | (95.28%)  |
|           | Val   | 9,992   | 0          | 0             | 0  | 4             | 32             | 14         | 2             | 157          | 9,783         | (97.91%)  |
|           | Test  | 9,993   | 0          | 0             | 0  | 7             | 33             | 8          | 1             | 59           | 9,885         | (98.92%)  |

Tab. 2: German text summarization datasets in numbers. Given are the original sample count and breakdown of filtered samples by automated assessment (cf., Sect. 3.1) for all provided splits. We set the *Minimum Length* to 20 characters for summaries and 50 for references, except for WikiLingua, which has limits of 8 and 20 characters, respectively, due to a different domain. *Id* refers to samples with same reference and summary text, *Min CR* ensures references are at least 25% longer than summaries, and *Fully Extr* identifies consecutive segments that are used as fully extractive summaries. For duplicates, we differentiate between both reference and summary appearing in the corpus (*Exact*), versus partial duplicates where only one of reference (*Ref*) or summary (*Summ*) are appearing elsewhere. Numbers in bold highlight issues affecting more than 2% of the split data.

directly running an extractive summarization system, and thus obtain similar (or better) quality at a much lower cost. For MassiveSumm, a large fraction of invalid samples can be attributed to duplicate content; manual inspection reveals that there are frequent generic references or summary texts, such as “*Read more after logging in!*”. We assume the reason to be a faulty extraction of HTML elements for particular websites.

The remaining inspected datasets were affected at a much lower rate; we see several subsets that have only a handful of faulty instances. Depending on the overall size of the dataset, this implies that evaluation scores will differ less between unfiltered and filtered splits of largely unaffected datasets.

## 4.2 Consistent Results and Baseline Runs

**Key Finding 2:** Existing evaluation scores are hard (if not outright impossible) to reproduce, even with model weights publicly available.

**Key Finding 3:** Authors frequently fail to put scores into context, not comparing their own results against baseline methods for further scrutiny.

Another worrying trend we observe in the “reproducibility” column of Tab. 1, is the consistent inability to even approximately reproduce self-reported scores for any of the evaluated

models. In our reproduction attempts, we employed no particular further filtering, and observed scores that were anywhere from 5 points worse to 3 points better than self-reported ones on the test set. Only a singular result was reproducible within 0.5 ROUGE points of the expected results. In particular, we find that implementation details on filtering steps and other subselection criteria are rarely (if ever) included in the documentation of training procedures. While the usage of so-called “model cards” [Mi19], i.e., dedicated documentation pages for particular training results, has improved the availability of at least *some form* of documentation, these descriptions are still insufficient to fully reproduce results. As a side note, it should also be mentioned that multiple implementations for the ROUGE evaluation metric exist<sup>16</sup>, which may result in scoring differences by utilizing different text processing tools or implementations. To ensure reproducibility of our own scores, we mention that scores were computed with help of the `rouge-score` package, version 0.1.2. We further replaced the default stemming algorithm with the German Cistem stemmer [WF17] to provide a reasonable upper-bound of scores and use the provided bootstrap sampler with  $n = 2000$ .

Aside from the lack of reproducible results, we also noted that only few public models report against a set of (consistent) baselines, with the most commonly compared approach being lead-3. Given that we have also presented a cleaned portion of popular evaluation models, we strive for a more comprehensive comparison of actual results, and investigate resulting implications that were omitted in the original evaluation settings.

In our particular setup, we compare against the three mentioned extractive baselines mentioned in Sect. 3.3 and report scores in Tab. 3. Depending on the dataset, the choice of a baseline can heavily skew the interpretation compared to neural methods. For example, on the Klexikon dataset, using lead-3 can lead to a roughly 12-13 point drop in ROUGE-1 scores compared to scores by the lead- $k$  or LexRank-ST baseline. On the other hand, for lead-heavy and short texts in MLSUM, lead-3 serves as the best baseline method. Our recommendation is therefore to similarly use multiple (different) baseline approaches, resulting in a more defined context for evaluation based on ROUGE scores. While it may be easier to simply copy results from prior work, we highly recommend the reproduction of these results first, as scores may ultimately vary between different experimental setups.

### 4.3 Impact of Data Filtering

**Key Finding 4:** After filtering, scores can drop by more than 20 ROUGE-1 points on the MLSUM test set.

To illustrate the effect of dataset filtering on downstream performance, we further compare results on the two most-affected datasets (MLSUM and MassiveSumm). Without any additional training, we run all available public models on the validation and test portion of

<sup>16</sup> e.g., `rouge-score` (<https://pypi.org/project/rouge-score/>, last accessed: 2022-10-06) or `pyrouge` (<https://pypi.org/project/pyrouge/>, last accessed: 2022-10-06)

| Dataset     | Method         | Validation Set |       |       | Test Set |       |       |
|-------------|----------------|----------------|-------|-------|----------|-------|-------|
|             |                | R-1            | R-2   | R-L   | R-1      | R-2   | R-L   |
| MLSUM       | lead-3         | 19.06          | 5.58  | 13.21 | 18.90    | 5.47  | 13.04 |
|             | lead- <i>k</i> | 14.93          | 4.12  | 11.31 | 15.08    | 4.17  | 11.45 |
|             | LexRank-ST     | 15.78          | 3.36  | 11.52 | 16.04    | 3.30  | 11.55 |
| Klexikon    | lead-3         | 15.19          | 3.46  | 9.10  | 15.87    | 3.64  | 9.35  |
|             | lead- <i>k</i> | 28.11          | 5.51  | 12.43 | 28.34    | 5.50  | 12.50 |
|             | LexRank-ST     | 27.23          | 4.63  | 11.48 | 27.42    | 4.58  | 11.55 |
| LegalSum    | lead-3         | 16.72          | 2.80  | 10.51 | 16.74    | 2.86  | 10.53 |
|             | lead- <i>k</i> | 14.34          | 2.27  | 8.78  | 14.36    | 2.34  | 8.78  |
|             | LexRank-ST     | 21.54          | 6.22  | 12.97 | 21.35    | 5.99  | 12.74 |
| EUR-Lex-Sum | lead-3         | 3.31           | 2.25  | 2.72  | 3.31     | 2.19  | 2.67  |
|             | lead- <i>k</i> | 41.74          | 17.77 | 16.04 | 39.42    | 17.08 | 15.52 |
|             | LexRank-ST     | 39.37          | 15.13 | 15.26 | 38.48    | 15.18 | 15.19 |

Tab. 3: Baseline results for all datasets with available validation and/or test splits. We report ROUGE F1 scores on the filtered datasets.

MLSUM, for which we also obtain scores on the original unfiltered sets. Our findings can be seen in Tab. 4, where one can observe a performance drop in *every model*, even those that were not originally trained on the MLSUM dataset itself (*t5-base*). By far the worst affected are the two baselines constructed from leading sentences, as well as the mT5-small models by users *mrm8488* and *Shahm*. These four models all achieve unreasonably high ROUGE-2 scores before filtering and see a reduction to about one fifth of the original scores after filtering. Upon inspection, we similarly found that these models were ultimately simply re-generating the first tokens from the input article. These findings are concerning, as they ultimately question the current state-of-the-art on the MLSUM dataset. It further validates the necessity of filtering, given that we can ultimately change the course of evaluation and interpretation of models. For MLSUM, per our results, the *t5-base* model, trained on a related news dataset and utilizing the largest underlying neural model, seems to perform best on filtered datasets while originally lagging behind even a simple lead-3 baseline. This is particularly interesting, because the underlying model checkpoint used is primarily trained on English texts.

Figure 2 further visualizes the impact of filtering on the length distributions of the two heavily affected datasets, MLSUM and MassiveSumm. Analyzing the resulting changes in more detail, we can observe a more strictly enforced minimum length for both references and summaries in the MLSUM dataset even before filtering. In stark contrast, MassiveSumm is shrunk considerably by the minimum length filter, which in turn shifts the samples towards generally longer reference texts. Since MLSUM is affected more by the extractiveness filter, one can observe a noticeable change in the mean of the distribution of summary texts, particularly longer ones.

Changes in the length distribution, however, do not explain any of the deterioration in raw ROUGE scores; a further indicator that several different evaluation methods need to be

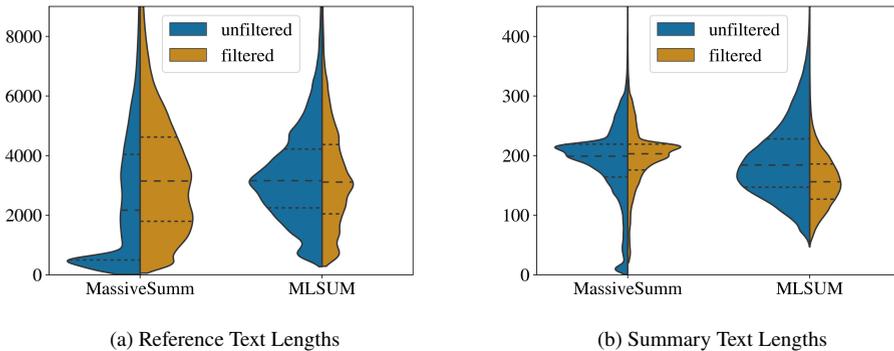


Fig. 2: Violin plots illustrating the distributional shift on the MLSUM and MassiveSumm training splits through filtering. Black dashed lines indicate mean and quartiles of the distribution.

combined in order to paint a more complete picture for the realistic performance of models. We particularly recommend the utilization of violin plots also for the evaluation of system outputs, as they allow the comparison of length estimates by the system in comparison to ground truth data.

#### 4.4 Qualitative Analysis of Generated Summaries

**Key Finding 5:** With the exception of one work [Ak20], no *publicly* available system performs experiments beyond simple ROUGE score computation.

**Key Finding 6:** Despite high reported scores, catastrophic failures can be observed in some systems.

**Key Finding 7:** All utilized architectures only work with a relatively limited context, proving to be incapable of dealing with long-form summarization.

The first criterion we were looking at when checking for existing systems is the evaluation setting that was used in the respective work. The findings, reported in Tab. 1, point towards a more rigorous evaluation setting for models backed by a scientific publication, which comes as no surprise. However, we also note that these systems are also more likely to withhold their respective models from public access. This ultimately means that those models can only be judged based on the reported evaluation and no further black-box model checks can be performed on them. To aggregate the insights gained across these works, most frequently mentioned is the issue of factual consistency [Ve19, FBZ19], which does not bode well for the practical suitability of such systems beyond simple settings. Secondly, several works also investigate system outputs’ fluency [FBZ19, Ak20], where abstractive models could provide sensible improvements over extractive systems. However, especially for earlier works, consistent generations from language models still prove to be difficult.

| Model             | MLSUM Validation Split |              |              |              |             |              | MLSUM Test Split |              |              |              |             |              |
|-------------------|------------------------|--------------|--------------|--------------|-------------|--------------|------------------|--------------|--------------|--------------|-------------|--------------|
|                   | Unfiltered             |              |              | Filtered     |             |              | Unfiltered       |              |              | Filtered     |             |              |
|                   | R-1                    | R-2          | R-L          | R-1          | R-2         | R-L          | R-1              | R-2          | R-L          | R-1          | R-2         | R-L          |
| <b>Lead-3</b>     | 36.22                  | 26.24        | 31.89        | 19.06        | 5.58        | 13.21        | 37.15            | 27.48        | 32.94        | 18.90        | 5.47        | 13.04        |
| <b>Lead-k</b>     | 29.25                  | 20.92        | 26.51        | 14.93        | 4.12        | 11.31        | 31.35            | 22.86        | 28.58        | 15.08        | 4.17        | 11.45        |
| <b>LexRank-ST</b> | 18.62                  | 6.46         | 14.26        | 15.78        | 3.36        | 11.52        | 18.83            | 6.45         | 14.36        | 16.04        | 3.30        | 11.55        |
| <b>mrm8488</b>    | <b>42.77</b>           | 31.89        | <b>38.93</b> | 21.63        | 6.64        | 16.32        | <b>44.05</b>     | 33.44        | <b>40.36</b> | 21.31        | 6.36        | 16.09        |
| <b>ml6team</b>    | 28.17                  | 18.81        | 26.05        | 17.08        | 5.03        | 14.18        | 28.51            | 19.52        | 26.53        | 16.56        | 4.80        | 13.78        |
| <b>T-Systems</b>  | 23.74                  | 11.08        | 20.34        | 19.87        | 6.49        | 16.40        | 23.67            | 11.21        | 20.36        | 19.20        | 6.11        | 15.84        |
| <b>Shahm</b>      | 42.59                  | <b>31.96</b> | 38.70        | 21.50        | 6.87        | 16.15        | 43.92            | <b>33.62</b> | 40.09        | 21.20        | 6.62        | 15.79        |
| <b>t5-base</b>    | 27.54                  | 11.31        | 20.88        | <b>23.31</b> | <b>7.19</b> | <b>16.99</b> | 27.99            | 11.65        | 21.20        | <b>23.40</b> | <b>7.20</b> | <b>16.91</b> |

Tab. 4: ROUGE F1 scores on the MLSUM validation and test splits, comparing results with and without data filtering. Across all tested models, a stark drop in performance can be observed. We highlight the highest score for each split in bold.

To follow our own advice, we manually investigated instances of generated outputs from systems in Table 4. In addition to samples from the MLSUM dataset, we further tested with instances from the Klexikon and WikiLingua datasets to check for domain generalization. As others have noted, the factual consistency of abstractive systems is questionable at best, but understated just how badly summaries can deviate from the original. Several times a reversed order of aggressors and victims (respectively, winners or losers in sports game) was generated, and in one particular instance the context was altered from “live-saving” to “drowning (someone)” by the summarization system. This happened on “in-domain samples” from the MLSUM test set.

A similar observation can be made for the syntactic quality of generations, where overfitting of systems becomes particularly apparent during the zero-shot evaluation on other datasets. While it can be expected that the quality of a generated summary may lack in content accuracy or truthfulness, oftentimes no coherent sentence was provided. Less tragic, but difficult for system comparison, is the multitude of parameters for generation functions. While self-reported scores of public models generally rely on greedily decoded summaries, one model frequently started repeating short sequences of about three words indefinitely until the maximum generation length was reached. Importantly, such repetitions are not obvious from looking at a ROUGE-based evaluation of model outputs alone, but could be easily suppressed by enabling  $n$ -gram-based filtering during the generation.

We were also able to verify that the highly-scoring models by users *mrm8488* and *Shahm* indeed only copy the leading tokens from the input samples, likely due to training on unfiltered MLSUM splits. This spells further trouble for “state-of-the-art” models, as it requires a deeper examination for determining which summaries are actually better than simple string selection approaches, such as lead-3. We hypothesize that the same concept used in our *extractiveness* filter can also be applied to generated outputs; with a slightly altered similarity scoring mechanism, e.g., the longest common subsequence algorithm, even near duplicates could be detected and flagged for manual review.

Most prominently though, due to architectural constraints of the underlying neural models, none of the currently public systems is able to capture an input context beyond 512 subword

tokens, the default length limit for the Transformer architectures [Va17]. In the instance of domain-specific datasets, such as EUR-Lex-Sum, this means that even the length of summary texts exceeds the limitation of models, effectively rendering them useless in this particular context.

## 5 Conclusion and Future Work

Studying the current landscape of German abstractive summarization initially paints a grim picture: While the general willingness and ease of sharing systems has greatly increased over the past years, around half of the currently known German summarization systems still remain inaccessible to the public. Of those that *are* available for public scrutiny, a prominent focus on news summarization is still persisting, preventing more broader applications. Even worse, the most prominent dataset contains severe flaws in the sample quality, leading to models whose generalization capabilities, even in-domain, are severely hampered by the unfiltered data. This also hints at the general level of care practitioners take with respect to exploratory data analysis, given that several issues can be spotted by simply inspecting just a few samples. And finally, even models that take care of filtering some of these issues, a qualitative analysis of generations can still reveal catastrophic problems that prevent an ethically responsible deployment of the solution in practice.

However, there are some silver linings at the horizon. Many of the major data-centric issues can be easily fixed with the introduced quality checks, which can be applied cost-effectively across multiple datasets, as we have demonstrated in this work. Through publishing our pre-processing pipeline, we hope to encourage others in taking a more data-centric exploration before starting with the ultimate model training.

Within just two years, we have also seen an unbelievable influx of available summarization datasets for German, importantly extending past the narrow domains into application-specific fields, such as law and medicine, and totaling more than 700.000 samples across publicly available resources. This hopefully paves the way towards a more consistent and generalized approach in German abstractive summarization research; should the efforts of the community keep at the current rate, we will likely see meaningful progress within the next year. The latest trends in the English summarization community also indicate a shift towards greater awareness of long-form summarization [PZL22]; while dedicated long context German (or multilingual) model checkpoints are still absent, we estimate that such systems will become available shortly, serving as a compute-intensive way to escape the current restrictions on input length.

As for our own efforts, we are currently investigating how systems can be designed to work well across multiple domains at once, without the need for several distinct models. This requires careful analysis of the underlying data, as well as a more agnostic training framework to prevent overfitting towards a particular style.

## Bibliography

- [ACG22] Aumiller, Dennis; Chouhan, Ashish; Gertz, Michael: EUR-Lex-Sum: A Multi- and Cross-lingual Dataset for Long-form Summarization in the Legal Domain. In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, pp. 7626–7639, December 2022.
- [AG22] Aumiller, Dennis; Gertz, Michael: Klexikon: A German Dataset for Joint Summarization and Simplification. In: Proceedings of the Language Resources and Evaluation Conference. European Language Resources Association, Marseille, France, pp. 2693–2701, June 2022.
- [Ak20] Aksenov, Dmitrii; Schneider, Julián Moreno; Bourgonje, Peter; Schwarzenberg, Robert; Hennig, Leonhard; Rehm, Georg: Abstractive Text Summarization based on Language Model Conditioning and Locality Modeling. In (Calzolari, Nicoletta; Béchet, Frédéric; Blache, Philippe; Choukri, Khalid; Cieri, Christopher; Declerck, Thierry; Goggi, Sara; Isahara, Hitoshi; Maegaard, Bente; Mariani, Joseph; Mazo, Hélène; Moreno, Asunción; Odijk, Jan; Piperidis, Stelios, eds): Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020. European Language Resources Association, pp. 6680–6689, 2020.
- [BC20] Bommasani, Rishi; Cardie, Claire: Intrinsic Evaluation of Summarization Datasets. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Online, pp. 8075–8096, November 2020.
- [BCB15] Bahdanau, Dzmitry; Cho, Kyunghyun; Bengio, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. In (Bengio, Yoshua; LeCun, Yann, eds): 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015.
- [Be16] Benikova, Darina; Mieskes, Margot; Meyer, Christian M.; Gurevych, Iryna: Bridging the gap between extractive and abstractive summaries: Creation and evaluation of coherent extracts from heterogeneous sources. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers. The COLING 2016 Organizing Committee, Osaka, Japan, pp. 1039–1050, December 2016.
- [Bi20] Biesner, David; Brito, Eduardo; Hillebrand, Lars Patrick; Sifa, Rafet: Hybrid Ensemble Predictor as Quality Metric for German Text Summarization: Fraunhofer IAIS at GermEval 2020 Task 3. In (Ebling, Sarah; Tuggener, Don; Hürlimann, Manuela; Cieliebak, Mark; Volk, Martin, eds): Proceedings of the 5th Swiss Text Analytics Conference and the 16th Conference on Natural Language Processing, SwissText/KONVENS 2020, Zurich, Switzerland, June 23-25, 2020. volume 2624 of CEUR Workshop Proceedings. CEUR-WS.org, 2020.
- [DBWR21] Deutsch, Daniel; Bedrax-Weiss, Tania; Roth, Dan: Towards Question-Answering as an Automatic Metric for Evaluating the Content Quality of a Summary. Transactions of the Association for Computational Linguistics, 9:774–789, 2021.
- [DDR21] Deutsch, Daniel; Dror, Rotem; Roth, Dan: A Statistical Analysis of Summarization Evaluation Metrics Using Resampling Methods. Transactions of the Association for Computational Linguistics, 9:1132–1146, 2021.

- [ECM19] Ermakova, Liana; Cossu, Jean Valère; Mothe, Josiane: A survey on evaluation of summarization methods. *Information processing & management*, 56(5):1794–1814, 2019.
- [ER04] Erkan, Günes; Radev, Dragomir R.: LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22:457–479, 2004.
- [Fa21] Fabbri, Alexander R.; Kryściński, Wojciech; McCann, Bryan; Xiong, Caiming; Socher, Richard; Radev, Dragomir: SummEval: Re-evaluating Summarization Evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409, 2021.
- [FBZ19] Fecht, Pascal; Blank, Sebastian; Zorn, Hans-Peter: Sequential Transfer Learning in NLP for German Text Summarization. In (Cieliebak, Mark; Tuggener, Don; Benites, Fernando, eds): *Proceedings of the 4th Swiss Text Analytics Conference, SwissText 2019*, Winterthur, Switzerland, June 18-19, 2019. volume 2458 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.
- [Fr20] Frefel, Dominik: Summarization Corpora of Wikipedia Articles. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. European Language Resources Association, Marseille, France, pp. 6651–6655, May 2020.
- [FVM20] Frefel, Dominik; Vogel, Manfred; Märki, Fabian: 2nd German Text Summarization Challenge. In (Ebling, Sarah; Tuggener, Don; Hürlimann, Manuela; Cieliebak, Mark; Volk, Martin, eds): *Proceedings of the 5th Swiss Text Analytics Conference and the 16th Conference on Natural Language Processing, SwissText/KONVENS 2020*, Zurich, Switzerland, June 23-25, 2020. volume 2624 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.
- [GMM21] Glaser, Ingo; Moser, Sebastian; Matthes, Florian: Summarization of German Court Rulings. In: *Proceedings of the Natural Legal Language Processing Workshop 2021*. Association for Computational Linguistics, Punta Cana, Dominican Republic, pp. 180–189, November 2021.
- [GNA18] Grusky, Max; Naaman, Mor; Artzi, Yoav: Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, pp. 708–719, June 2018.
- [IPM20] Iskender, Neslihan; Polzehl, Tim; Möller, Sebastian: Best Practices for Crowd-based Evaluation of German Summarization: Comparing Crowd, Expert and Automatic Evaluation. In: *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*. Association for Computational Linguistics, Online, pp. 164–175, November 2020.
- [JJB21] Johner, Timo; Jana, Abhik; Biemann, Chris: Error Analysis of using BART for Multi-Document Summarization: A Study for English and German Language. In: *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*. Linköping University Electronic Press, Sweden, Reykjavik, Iceland, pp. 391–397, May 31–2 June 2021.
- [Kr20] Kryscinski, Wojciech; McCann, Bryan; Xiong, Caiming; Socher, Richard: Evaluating the Factual Consistency of Abstractive Text Summarization. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, pp. 9332–9346, November 2020.

- [Kr22] Kreutzer, Julia; Caswell, Isaac; Wang, Lisa; Wahab, Ahsan; van Esch, Daan; Ulzii-Orshikh, Nasanbayar; Tapo, Allahsera; Subramani, Nishant; Sokolov, Artem; Sikasote, Claytone; Setyawan, Monang; Sarin, Supheakmungkol; Samb, Sokhar; Sagot, Benoît; Rivera, Clara; Rios, Annette; Papadimitriou, Isabel; Osei, Salomey; Suarez, Pedro Ortiz; Orife, Iroro; Ogueji, Kelechi; Rubungo, Andre Niyongabo; Nguyen, Toan Q.; Müller, Mathias; Müller, André; Muhammad, Shamsuddeen Hassan; Muhammad, Nanda; Mnyakeni, Ayanda; Mirzakhlov, Jamshidbek; Matangira, Tapiwanashe; Leong, Colin; Lawson, Nze; Kudugunta, Sneha; Jernite, Yacine; Jenny, Mathias; Firat, Orhan; Dossou, Bonaventure F. P.; Dlamini, Sakhile; de Silva, Nisansa; Çabuk Ballı, Sakine; Biderman, Stella; Battisti, Alessia; Baruwa, Ahmed; Bapna, Ankur; Baljekar, Pallavi; Azime, Israel Abebe; Awokoya, Ayodele; Ataman, Duygu; Ahia, Orevaoghene; Ahia, Oghenefego; Agrawal, Sweta; Adeyemi, Mofetoluwa: Quality at a Glance: An Audit of Web-Crawled Multilingual Datasets. *Transactions of the Association for Computational Linguistics*, 10:50–72, 2022.
- [La20] Ladhak, Faisal; Durmus, Esin; Cardie, Claire; McKeown, Kathleen: WikiLingua: A New Benchmark Dataset for Cross-Lingual Abstractive Summarization. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, pp. 4034–4048, November 2020.
- [Le20] Lewis, Mike; Liu, Yinhan; Goyal, Naman; Ghazvininejad, Marjan; Mohamed, Abdelrahman; Levy, Omer; Stoyanov, Veselin; Zettlemoyer, Luke: BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In (Jurafsky, Dan; Chai, Joyce; Schluter, Natalie; Tetreault, Joel R., eds): *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020*, Online, July 5-10, 2020. Association for Computational Linguistics, pp. 7871–7880, 2020.
- [Li04] Lin, Chin-Yew: ROUGE: A Package for Automatic Evaluation of Summaries. In: *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, pp. 74–81, July 2004.
- [Li20] Liu, Yinhan; Gu, Jiatao; Goyal, Naman; Li, Xian; Edunov, Sergey; Ghazvininejad, Marjan; Lewis, Mike; Zettlemoyer, Luke: Multilingual Denoising Pre-training for Neural Machine Translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.
- [Li22] Liang, Siting; Kades, Klaus; Fink, Matthias; Full, Peter; Weber, Tim; Kleesiek, Jens; Strube, Michael; Maier-Hein, Klaus: Fine-tuning BERT Models for Summarizing German Radiology Findings. In: *Proceedings of the 4th Clinical Natural Language Processing Workshop*. Association for Computational Linguistics, Seattle, WA, pp. 30–40, July 2022.
- [Ma22] May, Philip: , Anomalies in the MLSUM Dataset. <https://may.la/blog/2022/02/23/anomalies-in-the-mlsum-dataset/>, 2022. Accessed: 2023-01-11.
- [Mi19] Mitchell, Margaret; Wu, Simone; Zaldivar, Andrew; Barnes, Parker; Vasserman, Lucy; Hutchinson, Ben; Spitzer, Elena; Raji, Inioluwa Deborah; Gebru, Timnit: Model Cards for Model Reporting. In (danah boyd; Morgenstern, Jamie H., eds): *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT\* 2019*, Atlanta, GA, USA, January 29-31, 2019. ACM, pp. 220–229, 2019.

- [MT04] Mihalcea, Rada; Tarau, Paul: TextRank: Bringing Order into Text. In: Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Barcelona, Spain, pp. 404–411, July 2004.
- [Na21] Nan, Feng; Nogueira dos Santos, Cicero; Zhu, Henghui; Ng, Patrick; McKeown, Kathleen; Nallapati, Ramesh; Zhang, Dejjiao; Wang, Zhiguo; Arnold, Andrew O.; Xiang, Bing: Improving Factual Consistency of Abstractive Summarization via Question Answering. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Association for Computational Linguistics, Online, pp. 6881–6894, August 2021.
- [NCL18] Narayan, Shashi; Cohen, Shay B.; Lapata, Mirella: Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Brussels, Belgium, pp. 1797–1807, October–November 2018.
- [Ni19] Nitsche, Matthias: Towards German Abstractive Text Summarization using Deep Learning. Master's thesis, Hochschule für angewandte Wissenschaften Hamburg, 2019.
- [NZZ17] Nallapati, Ramesh; Zhai, Feifei; Zhou, Bowen: Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In: Proceedings of the AAAI Conference on Artificial Intelligence. volume 31, 2017.
- [PBL21] Perez-Beltrachini, Laura; Lapata, Mirella: Models and Datasets for Cross-Lingual Summarisation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Punta Cana, Dominican Republic, pp. 9408–9423, November 2021.
- [PC20] Paraschiv, Andrei; Cercel, Dumitru-Clementin: UPB at GermEval-2020 Task 3: Assessing Summaries for German Texts using BERTScore and Sentence-BERT. In (Ebling, Sarah; Tuggener, Don; Hürlimann, Manuela; Cieliebak, Mark; Volk, Martin, eds): Proceedings of the 5th Swiss Text Analytics Conference and the 16th Conference on Natural Language Processing, SwissText/KONVENS 2020, Zurich, Switzerland, June 23–25, 2020. volume 2624 of CEUR Workshop Proceedings. CEUR-WS.org, 2020.
- [PM19] Parida, Shantipriya; Motlíček, Petr: Idiap Abstract Text Summarization System for German Text Summarization Task. In (Cieliebak, Mark; Tuggener, Don; Benites, Fernando, eds): Proceedings of the 4th Swiss Text Analytics Conference, SwissText 2019, Winterthur, Switzerland, June 18–19, 2019. volume 2458 of CEUR Workshop Proceedings. CEUR-WS.org, 2019.
- [PZL22] Phang, Jason; Zhao, Yao; Liu, Peter J: Investigating Efficiently Extending Transformers for Long Input Summarization. arXiv preprint arXiv:2208.04347, 2022.
- [Ra20] Raffel, Colin; Shazeer, Noam; Roberts, Adam; Lee, Katherine; Narang, Sharan; Matena, Michael; Zhou, Yanqi; Li, Wei; Liu, Peter J.: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.
- [Re00] Reithinger, Norbert; Kipp, Michael; Engel, Ralf; Alexandersson, Jan: Summarizing Multilingual Spoken Negotiation Dialogues. In: Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Hong Kong, pp. 310–317, October 2000.

- [RG19] Reimers, Nils; Gurevych, Iryna: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Association for Computational Linguistics, Hong Kong, China, pp. 3982–3992, November 2019.
- [RG20] Reimers, Nils; Gurevych, Iryna: Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Online, pp. 4512–4525, November 2020.
- [Sc20] Scialom, Thomas; Dray, Paul-Alexis; Lamprier, Sylvain; Piwowarski, Benjamin; Staiano, Jacopo: MLSUM: The Multilingual Summarization Corpus. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Online, pp. 8051–8067, November 2020.
- [SDP20] Sellam, Thibault; Das, Dipanjan; Parikh, Ankur: BLEURT: Learning Robust Metrics for Text Generation. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, pp. 7881–7892, July 2020.
- [SJ97] Sparck Jones, Karen: Summarising: Where are we now? Where should we go? In: Intelligent Scalable Text Summarization. 1997.
- [SVL14] Sutskever, Ilya; Vinyals, Oriol; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In (Ghahramani, Zoubin; Welling, Max; Cortes, Corinna; Lawrence, Neil D.; Weinberger, Kilian Q., eds): Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. pp. 3104–3112, 2014.
- [tHKdR20] ter Hoeve, Maartje; Kiseleva, Julia; de Rijke, Maarten: What Makes a Good Summary? Reconsidering the Focus of Automatic Summarization. CoRR, abs/2012.07619, 2020.
- [Ur22] Urlana, Ashok; Surange, Nirmal; Baswani, Pavan; Ravva, Priyanka; Shrivastava, Manish: TeSum: Human-Generated Abstractive Summarization Corpus for Telugu. In: Proceedings of the Language Resources and Evaluation Conference. European Language Resources Association, Marseille, France, pp. 5712–5722, June 2022.
- [Va17] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz; Polosukhin, Illia: Attention is All you Need. In (Guyon, Isabelle; von Luxburg, Ulrike; Bengio, Samy; Wallach, Hanna M.; Fergus, Rob; Vishwanathan, S. V. N.; Garnett, Roman, eds): Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 5998–6008, 2017.
- [Ve19] Venzin, Valentin; Deriu, Jan; Orel, Didier; Cieliebak, Mark: Fact-aware Abstractive Text Summarization using a Pointer-Generator Network. In (Cieliebak, Mark; Tuggener, Don; Benites, Fernando, eds): Proceedings of the 4th Swiss Text Analytics Conference, SwissText 2019, Winterthur, Switzerland, June 18-19, 2019. volume 2458 of CEUR Workshop Proceedings. CEUR-WS.org, 2019.
- [VS21] Varab, Daniel; Schluter, Natalie: MassiveSumm: a very large-scale, very multilingual, news summarisation dataset. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Punta Cana, Dominican Republic, pp. 10150–10161, November 2021.

- [WF17] Weissweiler, Leonie; Fraser, Alexander: Developing a Stemmer for German Based on a Comparative Analysis of Publicly Available Stemmers. In (Rehm, Georg; Declerck, Thierry, eds): *Language Technologies for the Challenges of the Digital Age - 27th International Conference, GSCL 2017, Berlin, Germany, September 13-14, 2017, Proceedings*. volume 10713 of *Lecture Notes in Computer Science*. Springer, pp. 81–94, 2017.
- [Wo20] Wolf, Thomas; Debut, Lysandre; Sanh, Victor; Chaumond, Julien; Delangue, Clement; Moi, Anthony; Cistac, Pierrick; Rault, Tim; Louf, Remi; Funtowicz, Morgan; Davison, Joe; Shleifer, Sam; von Platen, Patrick; Ma, Clara; Jernite, Yacine; Plu, Julien; Xu, Canwen; Le Scao, Teven; Gugger, Sylvain; Drame, Mariama; Lhoest, Quentin; Rush, Alexander: *Transformers: State-of-the-Art Natural Language Processing*. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, pp. 38–45, October 2020.
- [Xu21] Xue, Linting; Constant, Noah; Roberts, Adam; Kale, Mihir; Al-Rfou, Rami; Siddhant, Aditya; Barua, Aditya; Raffel, Colin: mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. In (Toutanova, Kristina; Rumshisky, Anna; Zettlemoyer, Luke; Hakkani-Tür, Dilek; Beltagy, Iz; Bethard, Steven; Cotterell, Ryan; Chakraborty, Tanmoy; Zhou, Yichao, eds): *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Association for Computational Linguistics, pp. 483–498, 2021.
- [Zh19] Zhong, Ming; Wang, Danqing; Liu, Pengfei; Qiu, Xipeng; Huang, Xuanjing: A Closer Look at Data Bias in Neural Extractive Summarization Models. In: *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Association for Computational Linguistics, Hong Kong, China, pp. 80–89, November 2019.
- [Zh20] Zhang, Tianyi; Kishore, Varsha; Wu, Felix; Weinberger, Kilian Q.; Artzi, Yoav: BERTScore: Evaluating Text Generation with BERT. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
- [Zh21a] Zhu, Chenguang; Hinthorn, William; Xu, Ruochen; Zeng, Qingkai; Zeng, Michael; Huang, Xuedong; Jiang, Meng: Enhancing Factual Consistency of Abstractive Summarization. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, pp. 718–733, June 2021.
- [Zh21b] Zhu, Chenguang; Yang, Ziyi; Gmyr, Robert; Zeng, Michael; Huang, Xuedong: Leveraging Lead Bias for Zero-shot Abstractive News Summarization. In (Diaz, Fernando; Shah, Chirag; Suel, Torsten; Castells, Pablo; Jones, Rosie; Sakai, Tetsuya, eds): *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*. ACM, pp. 1462–1471, 2021.
- [Zo18] Zopf, Markus: Auto-hMDS: Automatic Construction of a Large Heterogeneous Multilingual Multi-Document Summarization Corpus. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA), Miyazaki, Japan, May 2018.

# Detection of Generated Text Reviews by Leveraging Methods from Authorship Attribution: Predictive Performance vs. Resourcefulness

Manfred Moosleitner<sup>1</sup>, Günther Specht<sup>1</sup>, Eva Zangerle<sup>1</sup>

**Abstract:** Textual reviews are an integral part of online shopping, provided the reviews are authentic. To this end, pre-trained large language models have been shown to generate convincing text reviews at scale. Therefore, a critical task is the automatic detection of reviews not composed by humans. State-of-the-art approaches to detect generated texts use pre-trained large language models, which exhibit hefty hardware requirements to run and fine-tune. Previous work has shown that texts generated by the same language model show a coherent writing style. We propose to leverage this property to identify whether a text was indeed automatically generated. In this paper, we investigate the performance of features prominently used for authorship attribution, using classifiers with substantially lower computational resource requirements. We show that features and methods from authorship attribution can be successfully applied for the task of detecting generated text reviews, leveraging the consistent writing style exhibited by large language models like GPT-2. We argue that our approach achieves similar performance as state-of-the-art approaches while providing shorter training times and lower hardware requirements, necessary for, e.g., ad-hoc detection tasks.

**Keywords:** Text Classification, Stylometric Text Features, Generated Text Detection

## 1 Introduction

User-created reviews are often available on online e-commerce platforms. Such reviews allow users to express their satisfaction or disappointment with products or services in the form of numeric ratings or written text reviews. While user ratings provide a quantitative view of user experiences, text reviews allow for providing a more detailed report about the perceived quality of the product or service. Such reviews may inform other users in the process of comparing products, finding viable alternatives, or eventually, purchasing a product. However, this assumes that reviews are authentic and not fictitious and therefore, fake. Fake reviews can be a threat to businesses [La12, LSV16, LZ16], regardless of whether a counterfeit review is written by a human or generated with the help of an algorithm.

Recently, machine learning approaches like pre-trained large language models (LLM), such as BERT [De18] or GPT-2 [Ra19], are prevalent in many natural language processing and text generation tasks. To this end, GPT-2 was used to generate convincing text reviews [Ip20, Sa22], substantiating that we need to find ways to differentiate between human-written and algorithmically generated texts. Ott et al. [Ot11] investigated differentiating

---

<sup>1</sup> Universität Innsbruck, Department of Computer Science, Austria; [firstname.lastname@uibk.ac.at](mailto:firstname.lastname@uibk.ac.at)

between genuine and fabricated human-written reviews based on psycho-linguistic, lexical and  $n$ -gram features. They used these features and trained a Naïve Bayes and a linear Support Vector Machine as classifiers. Notably, in their evaluation, they also show that automated classification achieves better results than human judges. Ippolito et al. [Ip20] show that a large BERT model, fine-tuned for classification, can classify texts as either machine-generated or written by a human with an accuracy of up to 0.88, but also note less diversity for words used, due to preferably choosing words with a higher likelihood from the prevailing word distribution. Shahid et al. [Sh17] argue that if texts are generated by the same algorithm, these texts share the same author. Along these lines, we propose formulating the task of fake review detection as an authorship attribution task. Therefore, we leverage stylometric text features prominently used in the field of authorship attribution [Z118, St21, TMS19, MS22] to detect text reviews that are generated by an LLM. Our contributions can be summarized as follows: (1) We propose modeling the task of detecting generated text reviews as an authorship attribution task. (2) We investigate and compare the performance of different stylometric text features and state-of-the-art authorship attribution approaches. (3) We show that Support Vector Machines and Decision Trees achieve predictive performance ( $F_1$ , precision, recall) comparable to those of LLM-based classifiers while having much lower requirements in terms of training time and hardware requirements. Particularly given the recent trend towards greener IT and more energy-efficient computing [Mu08], we argue that this is a pivotal dimension that needs to be considered when evaluating and comparing potential approaches. (4) To ensure reproducibility, we publish the code of our experiments and analysis at [https://git.uibk.ac.at/c7031305/btw23\\_textreviewdetection](https://git.uibk.ac.at/c7031305/btw23_textreviewdetection).

## 2 Related Work

Ott et al. [Ot11] used part-of-speech (POS) tags, psycholinguistic and statistical text features, and  $n$ -grams to differentiate if human written reviews are genuine or fictive. They used 400 genuine reviews from Tripadvisor and 400 fabricated reviews, created by crowd workers. The authors used classifiers based on Naive Bayes, and on Support Vector Machines (SVM). Shahid et al. [Sh17] aim to separate Wikipedia articles and texts generated by content-spinning tools, using said articles as seed documents, by using an SVM-based classifier. They employed an assortment of stylometric features like  $n$ -grams, vocabulary richness, readability, and others. Salminen et al. [Sa22] evaluated the predictive performance of a RoBERTa [Li19] based model, a GPT-2 based model<sup>2</sup>, and an SVM based classifier. They created a balanced data set, consisting of approximately 40,000 text reviews, based on the Amazon Customer Review data set<sup>3</sup>. From the ten most occurring product categories in this data set, they sampled customer reviews to fine-tune an LLM to generate approximately 20,000 artificial text reviews. The authors additionally drew roughly 20,000 reviews from the Amazon data set as reviews written by humans.

---

<sup>2</sup> <https://github.com/openai/gpt-2-output-dataset/blob/master/detection.md>

<sup>3</sup> <https://s3.amazonaws.com/amazon-reviews-pds/readme.html>

Shahid et al. [Sh17] note that the seed documents and the generated texts differ in the way the texts are composed (i.e., their grammatical structure). Therefore, methods and features from the field of authorship attribution and plagiarism detection should be able to catch these differences. Zlatkova et al. [Z118] proposed to use various frequencies on word and sentence level, lexical richness, readability metrics, and other features. They reached first place in the multi-author analysis shared challenge at the PAN<sup>4</sup> workshop in 2018. Strøm [St21] used a similar configuration for this challenge in 2021, reaching high performance in this authorship attribution task. Murauer et al. [MS22] proposed Dependency Tree-grams (DT-grams) for the task of authorship attribution. DT-grams capture the writing style of authors by using the grammatical structure of sentences. Substructures, extracted from the dependency trees, are used to represent the grammatical style of the text and author.

In this work, we put our focus on identifying whether a review was written by a human, or was generated by an LLM. Along the lines of Shahid et al. [Sh17], we argue that texts originating from the same language model share the same author, which in turn should exhibit a similar writing style across all generated texts. Therefore, contrary to previous works, we model the task of detecting whether a review was manually written or automatically generated as an authorship attribution problem. We particularly investigate the use of state-of-the-art authorship attribution models to the task and particularly investigate the feature sets by Zlatkova [Z118]/Strøm [St21], and Murauer et al. [MS22].

### 3 Methodology

In the following, we first detail the different features employed and subsequently, describe the experimental setup used for the evaluation.

#### 3.1 Features

The main goal of this work is to investigate generated review detection by leveraging features and models from the field of authorship attribution. Therefore, we rely on features that have been shown to capture the writing style of authors well for authorship attribution tasks, specifically in the sub-tasks of style change detection [Z118, St21], intrinsic plagiarism detection [TMS19], and cross-language authorship attribution [MS22], leveraging the consistent writing style exhibited by texts generated by LLMs [Sh17, Ip20]. As baselines, we rely on word and character n-grams for comparison, along the lines of previous work [JL08, Ot11, Sh17]. We propose three types of features to represent reviews: (1) *textfeatures* (frequencies on word, phrase, and sentence level, and readability metrics); (2) *dtgrams* (substructures extracted from dependency trees); and (3) *ngrams* (word and character n-grams of varying lengths).

---

<sup>4</sup> <https://pan.webis.de/shared-tasks.html>

As *textfeatures*, we use the same collection of features as Zlatkova et al. [Z118] and Strøm [St21] to represent the writing styles of the corresponding authors. Our implementation is based on the work by Strøm<sup>5</sup>, in which the following five feature sets are extracted from the text reviews: (1) count metrics (for instance, number of sentences and words, count of English POS tags, capitalized words, and others); (2) number of occurrences of function words and function phrases; (3) the number of uses of digits (0, 1, ..., 9) and their alphabetical counterpart (zero, one, ..., nine), use of UK English (e.g., colour) and US English words (e.g., color), and the use of contractions and non-contractions; (4) the frequency of punctuation marks and other special characters; and (5) nine readability metrics (e.g., Flesch reading ease [Fl48]). This provides us with a total number of 487 features. For the *dtgrams* features, we rely on dependency-tree-based features, aiming to find texts that are composed with a similar grammatical style, therefore, attributing it to the generating LLM as the single author. In the following, we briefly introduce DT-grams. At first, we create the dependency tree for a given sentence (cf. Figure 1 for an example tree of the sentence “The quick brown fox jumped over the lazy dog.”). In the resulting tree, each node holds the English POS tag for the corresponding word. Next, specific substructures of the tree are grouped together, where the substructures can be of different shapes. For our work, we used the shape of *pq-grams* with  $p = 2$  and  $q = 3$ , which means that we use the parent-child relation of two nodes (“jumped” and “dog” as one example from Figure 1) and three sibling nodes (“over”, “the”, and “lazy” as one example from Figure 1). In the next step, the DT-grams are constructed using a sliding window, similar to n-grams. Here the grouped substructures are traversed (from parent to child, and siblings from left to right) and their corresponding POS tags are concatenated using the underscore as delimiter and the asterisk for when nodes are absent in the sliding window. The sentence in the given example results in a total of 18 strings, with “VBD\_NN\_IN\_DT\_JJ” being an example with no absent node and “VBD\_NN\_\*\_\*\_IN” being an example with absent nodes.

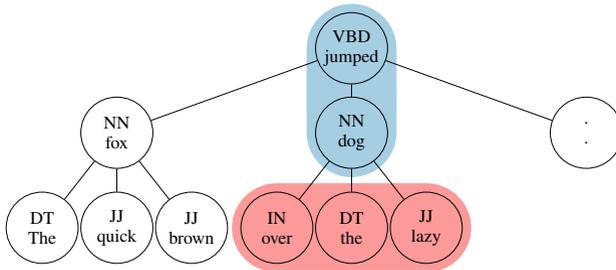


Fig. 1: Tree representation for the given sentence, based on the dependency tree. In each node, we display the word and its corresponding POS tags. Highlighted in blue is the parent-child relationship between *jumped* and *dog*, and marked in red is the sibling relationship between *over*, *the*, and *lazy*. Here, “VBD\_NN\_IN\_DT\_JJ” and “VBD\_NN\_\*\_\*\_IN” are two examples of the 18 dtgrams.

For the calculation of the dt-grams, we used two Python libraries from Murauer et al. [MS22]:

<sup>5</sup> <https://github.com/eivistr/pan21-style-change-detection-stacking-ensemble>

*tuhlbox*<sup>6</sup> parses the texts and calculates the dependency trees using the Python library Stanza [Qi20] from the Stanford NLP group, and *treegrams*<sup>7</sup>, is used to extract the dt-grams from these dependency trees. For the *n-grams* feature set and as a baseline, we employ word and character n-gram TF-IDF vectors, using the scikit-learn library<sup>8</sup>. We utilized similar parameters for the vectorization as [Z118, St21], further details are given in Section 3.2.

### 3.2 Experimental Setup

We evaluated the classification performance of the proposed approach in multiple experiments. Specifically, we employed stratified ten-fold cross-validation and used  $F_1$ , precision, and recall, as evaluation metrics. For the binary classification, the generated reviews were used as the positive class. Furthermore, we also measured classification runtimes and memory usage to compare the efficiency and required resources for each of the feature sets and classifiers. Memory usage was recorded using the Python library memory-profiler; the experiments were executed on a general-purpose processor.

#### 3.2.1 Dataset

We relied on the data set provided by Salminen et al. [Sa22] for the evaluations. The data set is balanced in the amount of original and generated text reviews, featuring approximately 20,000 text reviews per class. We computed all features proposed and normalized them by removing the mean and scaling the values to unit variance (using scikit-learn’s StandardScaler).

| rating | class | text  |
|--------|-------|---|
| 1      | CG    | Editor was too busy watching “Duck Dynasty” and not paying attention to his work!!!                           |
| 5      | CG    | I loved this book. The characters were believable and the plot was interesting.<br>I really enjoyed this book |
| 1      | OR    | Just a bit strange and different for me. Probably excellent for others.                                       |
| 5      | OR    | A truly riveting page turner. All three in the series were fantastic.   |

Tab. 1: Example reviews from the category Book\_5 for the classes Computer Generated (CG) and Original Review (OR), one with the highest and lowest rating each.

#### 3.2.2 Classification Algorithms and Evaluation Strategy

For the classification, we rely on five established classifiers: Support Vector Machine (SVM) with linear kernel, SVM with Radial Basis Function kernel, k-Nearest Neighbor (kNN),

<sup>6</sup> <https://pypi.org/project/tuhlbox/>

<sup>7</sup> <https://pypi.org/project/treegrams/>

<sup>8</sup> <https://scikit-learn.org/>

Gradient Boosting Decision Tree (gbdt), and Random Forest (rf). For the two SVM-based and the kNN-based classifiers, we used the corresponding scikit-learn implementations. For the tree-based classifiers, we utilized the gradient boosting framework LightGBM<sup>9</sup>. Following the example of Strøm[St21], we also used the hyperparameter optimization framework Optuna<sup>10</sup> to efficiently tune the hyperparameters for gbdt. The hyperparameters for the SVM-based classifiers were optimized using a grid-search approach.

This optimized hyperparameters<sup>11</sup> were then used for the final set of experiments, where all the necessary data and results were collected. Since we used the data set from Salminen et al. [Sa22], we reproduced their setup using a basic RoBERTa [Li19] model and added it as a state-of-the-art baseline to our experiments. As a more lightweight LLM, we added a DistilBERT [Sa19] model to the list of classifiers.

Furthermore, we tested the values for  $F_1$ , precision, and recall, per ten-fold cross-validation, for normal distribution by performing a Shapiro-Wilk [SW65] test. This showed  $p > .05$  for all cases, therefore, we assume that the determined results all stem from a normal distribution. This allowed us to perform statistical significance tests using paired t-tests, where the pairs were built within feature set and also within classifier.

### 3.2.3 Preliminary Experiments

In preliminary experiments, we conducted a coarse grid search for the classifiers. Regarding the feature set ngrams, the texts were used as input without any further pre-processing. The following parameters were supplied with the stated values: maximum number of n-grams (5,000, 25,000, no limit), word and character n-grams, range of n-grams (uni- to six-grams). The classifier hyperparameters used for the grid search are shown in Table 2.

|            |   |
|------------|---|
| linear SVM | C: 0.1, 1.0, 10; dual: False; tol: 0.001  |
| rbf SVM    | C: 0.1, 1.0, 10; kernel: rbf; tol: 0.001  |
| kNN        | n_neighbors: 5, 40, 100   |
| gbdt & rf  | objective: binary<br>learning_rate: 0.1, 0.01, 0.001, 0.0001, 0.00001<br>bagging_freq: 40; bagging_fraction: 0.85 |

Tab. 2: Hyperparameters used for the initial grid search.

These experiments showed that the linear SVM, the SVM with a radial basis function kernel, and the Gradient-boosted decision tree constantly outperformed kNN and Random Forest classifiers. Therefore, we excluded kNN and Random Forest classifiers from further experiments.

<sup>9</sup> <https://github.com/microsoft/LightGBM>

<sup>10</sup> <https://optuna.org/>

<sup>11</sup> Details can be found at [https://git.uibk.ac.at/c7031305/btw23\\_textreviewdetection](https://git.uibk.ac.at/c7031305/btw23_textreviewdetection)

Likewise, we also experimented with different ranges for n-grams, using word and/or character n-grams, and the maximum number of features. Again, we performed a grid-search approach to get the individual  $F_1$ -scores for word and character uni- to six-grams, varying values of 5,000, 25,000, 50,000, and unlimited number of maximum features. The results have shown that character four-, five-, and six-grams performed best with all three classifiers. Regarding the maximum number of features, the SVM with radial basis function kernel performed best with a limit of 25,000 features, while the linear SVM and the Gradient-boosted decision tree performed best without limiting the number of features.

Based on the findings of these preliminary experiments, we performed a final optimization of the hyperparameters for the classification algorithms. Both SVM variants were tuned again using a grid-search approach, to find the best-performing value for the regularization parameter. For the SVM with radial basis function kernel, a value of 10 showed the best performance across the three feature sets. Regarding the linear SVM, a value 0.1 performed best for the feature set dtgrams, and 0.001 for the feature sets textfeatures and ngrams.

## 4 Experimental Results

Based on the collected predictions and measurements from our experiments, we compared the performance of the combinations of the feature sets with classifiers, in terms of prediction, runtime, and memory consumption.

### 4.1 Predictive Performance

We present the average  $F_1$ , precision, and recall from the stratified ten-fold cross-validations per feature set for the three classification algorithms and the two LLMs in Table 3.

For the textfeatures feature set, the gradient-boosted decision tree achieved the highest performance with an  $F_1$ -score of 90.83%, followed by the SVM with radial basis function kernel and the linear SVM with a slightly lower performance (89.46% and 89.04%, respectively). The best performance for the feature set dtgrams was achieved by the SVM with radial basis function kernel with an  $F_1$ -score of 91.86%. Here, the  $F_1$ -score of the linear SVM achieved a 1.85% and the Gradient-boosted decision tree a 2.82% lower  $F_1$ -score. For the ngrams feature set, the SVM with radial basis function kernel obtained an  $F_1$ -score of 95.45%. This outperformed the  $F_1$ -scores of the other two classifiers, with the linear SVM achieving a 1.35% and the gradient-boosted decision tree a 2.16% lower  $F_1$ -score.

The results of the paired t-tests within the evaluated feature sets and classifiers showed  $p < .05$  for all pairs. The paired t-test with the results of RoBERTa and DistilBERT also showed  $p < .05$ . We also compared the results of the best non-LLM model with the RoBERTa model in a paired t-test, which also showed significant differences ( $p < .05$ ).

|                |              | linear SVM    |          | SVM rbf       |          | GBDT   |          |
|----------------|--------------|---------------|----------|---------------|----------|--------|----------|
|                |              | $\mu$         | $\sigma$ | $\mu$         | $\sigma$ | $\mu$  | $\sigma$ |
| F <sub>1</sub> | textfeatures | 0.8904        | 0.0037   | 0.8946        | 0.0044   | 0.9083 | 0.0027   |
|                | dtgrams      | 0.9028        | 0.0036   | 0.9186        | 0.0042   | 0.8904 | 0.0046   |
|                | ngrams       | 0.9410        | 0.0039   | <b>0.9545</b> | 0.0019   | 0.9329 | 0.0039   |
|                | RoBERTa      | <b>0.9794</b> | 0.0027   |               |          |        |          |
|                | DistilBERT   | 0.9670        | 0.0035   |               |          |        |          |
| PRECISION      | textfeatures | 0.9011        | 0.0050   | 0.8910        | 0.0056   | 0.9078 | 0.0053   |
|                | dtgrams      | 0.8765        | 0.0044   | 0.8998        | 0.0054   | 0.8832 | 0.0060   |
|                | ngrams       | <b>0.9639</b> | 0.0034   | 0.9446        | 0.0020   | 0.9244 | 0.0050   |
|                | RoBERTa      | <b>0.9946</b> | 0.0015   |               |          |        |          |
|                | DistilBERT   | 0.9882        | 0.0022   |               |          |        |          |
| RECALL         | textfeatures | 0.8799        | 0.0053   | 0.8982        | 0.0053   | 0.9089 | 0.0037   |
|                | dtgrams      | 0.9308        | 0.0066   | 0.9384        | 0.0052   | 0.8977 | 0.0091   |
|                | ngrams       | 0.9193        | 0.0078   | <b>0.9646</b> | 0.0039   | 0.9415 | 0.0054   |
|                | RoBERTa      | <b>0.9647</b> | 0.0055   |               |          |        |          |
|                | DistilBERT   | 0.9466        | 0.0065   |               |          |        |          |

Tab. 3: Mean and standard deviation for F<sub>1</sub>, precision, and recall over the values from the ten-fold cross-validation. Results are reported per classifier per feature set. The best results for the non-LLM and LLM approaches are marked in boldface.

From these results, we conclude that RoBERTa, as expected, achieves the best F<sub>1</sub>-scores regarding the predictive performance. The proposed text features, on the other hand, achieve only slightly lower F<sub>1</sub>-scores, with the best classifier achieving a less than 2% lower F<sub>1</sub>-score. However, we argue that the differences are subtle and, as we will show in the following experiments, the non-LLM approaches are able to outperform RoBERTa w.r.t. resource consumption and runtime.

## 4.2 Runtime Performance

Given the ever-increasing need for more energy-efficient computation, another important aspect is the runtime performance of the different approaches. We analyzed the time needed to fit the algorithms on average for the ten-fold cross-validation. All the experiments regarding runtime were conducted on the same virtual machine (Debian GNU/Linux 10 (buster) OS with 12 cores and 128GiB of memory). We utilize the grid-search functionality of scikit-learn. We present the average time to fit a classifier and the corresponding standard deviation in Table 4. The reported runtimes for ngrams, RoBERTa, and DistilBERT, include the time to convert the text reviews into their respective representations. The feature sets textfeatures and dtgrams were pre-computed, with a runtime of  $\approx 380$  seconds for the former and 73439.468 seconds for the latter.

The time to fit the models increases with the number of features, with the exception of the linear SVM when trained with ngrams. This constellation was about 2.4 times faster than

|              | linear SVM     |          | SVM rbf    |          | GBDT          |          |
|--------------|----------------|----------|------------|----------|---------------|----------|
|              | $\mu$          | $\sigma$ | $\mu$      | $\sigma$ | $\mu$         | $\sigma$ |
| textfeatures | 357.69s        | 54.09s   | 1,259.42s  | 107.04s  | <b>54.59s</b> | 2.17s    |
| dtgrams      | <b>490.26s</b> | 52.40s   | 4,020.95s  | 145.32s  | 2,135.30s     | 145.16s  |
| ngrams       | <b>146.29s</b> | 26.00s   | 17,605.94s | 439.07s  | 9,695.23s     | 137.98s  |
| RoBERTa      | 28149.44s      | 243.10s  |            |          |               |          |
| DistilBERT   | 8293,96s       | 104.49s  |            |          |               |          |

Tab. 4: Average ( $\mu$ ) and standard deviation ( $\sigma$ ) of the measured runtimes in seconds per classifier per feature set over the ten-fold cross-validation. The best values per feature set are marked in boldface.

when paired with textfeatures, and about 3.3 times faster when using dtgrams. Most notable are the runtimes for the gradient-boosting decision tree with textfeatures, the SVM with radial basis function kernel and ngrams, and RoBERTa. The first two displayed the shortest and the longest mean time to fit, respectively, from the non-LLM models, and RoBERTa the overall longest average time to fit. These experiments showed that the proposed simple approaches clearly outperform RoBERTa.

### 4.3 Memory Usage

Besides runtime, a second important cornerstone when evaluating and comparing these methods is their memory usage and hence, resource consumption. Particularly with LLMs, the amount of memory required has increased substantially. Therefore, we are also interested in comparing memory usage among the proposed approaches. One run of the final experiments was used to measure the amount of memory that was allocated during the execution of the cross-validation (cf. 4). We present the results of these analyses in Table 5. As expected, the memory profiler reported memory usage numbers for the non-LLM classifiers that are only a fraction of the memory needed by the RoBERTa model. Particularly, the recorded memory requirements of RoBERTa are in the range of 245.62 to 778.79 times higher compared to the non-LLM models with the lowest memory consumption.

|              | SVM linear     | SVM rbf    | GBDT       |
|--------------|----------------|------------|------------|
| textfeatures | 42.516 MiB     | 17.027 MiB | 19.520 MiB |
| dtgrams      | 17.301 MiB     | 16.961 MiB | 14.152 MiB |
| ngrams       | 13.312 MiB     | 19.543 MiB | 13.910 MiB |
| RoBERTa      | 10,364.988 MiB |            |            |
| DistilBERT   | 4,880.535 MiB  |            |            |

Tab. 5: Memory usage was acquired with cross-validation using the memory profiler.

## 4.4 Discussion

RoBERTa achieved the highest  $F_1$ -scores, followed by ngrams with rbfsvm, and ngrams with linsvm. When comparing these three, the highest  $F_1$ -score comes at the price of 1.6 and 192.2 times longer training times, for a difference of 1.8% and 3.15% in  $F_1$ -score, respectively. In terms of memory requirements, RoBERTa's memory usage is about 530 and 778 times higher when compared to rbfsvm with ngrams and linsvm with ngrams. In comparison, textfeatures with gbdt showed the shortest training time. Compared to RoBERTa, training times of ngrams with rbfsvm, ngrams with linsvm, and textfeatures with gbdt, were around 515, 322, and 3 times faster, and with a difference in  $F_1$ -scores of 6.42%, 4.62%, and 3.27%, respectively. The lowest memory requirement was recorded for ngrams with linsvm, which was 779 and 1.47 times lower than RoBERTa and ngrams with rbfsvm, with a difference in  $F_1$ -score of 3.15% and 1.35%.

Our experiments show it is possible to train classifiers quicker or with lower hardware requirements while sacrificing at most 6.42% of  $F_1$ -score, which is still higher than the performance of human raters [Ot11, Sa22]. Therefore, we argue that the proposed features borrowed from authorship attribution tasks are a valid option for the task of generated text detection.

## 5 Conclusion

We proposed using text features borrowed from the field of authorship attribution for the task of detecting generated product reviews. Related work suggests that generated texts differ in writing style, grammatical structure, and the diversity of words used from their input texts. Therefore, we used statistical textfeatures, features based on dependency-tree-grams, and  $n$ -grams to train a linear Support Vector Machine, a Support Vector Machine with radial basis function kernel, and a gradient-boosting decision tree as classifiers. We utilized a balanced dataset to evaluate their predictive performance using  $F_1$ -score, investigated their runtimes and memory requirements, also in comparison with a state-of-the-art RoBERTa LLM model. Our results show that classification algorithms like Support Vector Machines and Decision Trees can be trained using different stylometric features and achieve  $F_1$ -scores that come close to the performance of a basic RoBERTa model. While these non-LLM models show slightly lower performance, they can reach up to 515 faster training time and need up to 779 times less memory—two factors that become more and more central when choosing an approach.

In future work, we aim to investigate the importance and impact of individual features and extend the experiments to further datasets and text domains.

## Bibliography

- [De18] Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805, 2018.
- [Fl48] Flesch, Rudolph: A new readability yardstick. *Journal of applied psychology*, 32(3):221, 1948.
- [Ip20] Ippolito, Daphne; Duckworth, Daniel; Callison-Burch, Chris; Eck, Douglas: Automatic Detection of Generated Text is Easiest when Humans are Fooled. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, pp. 1808–1822, July 2020.
- [JL08] Jindal, Nitin; Liu, Bing: Opinion Spam and Analysis. In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*. WSDM '08, Association for Computing Machinery, New York, NY, USA, p. 219–230, 2008.
- [La12] Lappas, Theodoros: Fake Reviews: The Malicious Perspective. In (Bouma, Gosse; Ittoo, Ashwin; Métais, Elisabeth; Wortmann, Hans, eds): *Natural Language Processing and Information Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 23–34, 2012.
- [Li19] Liu, Yinhan; Ott, Myle; Goyal, Naman; Du, Jingfei; Joshi, Mandar; Chen, Danqi; Levy, Omer; Lewis, Mike; Zettlemoyer, Luke; Stoyanov, Veselin: RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv preprint arXiv:1907.11692, 2019.
- [LSV16] Lappas, Theodoros; Sabnis, Gaurav; Valkanas, Georgios: The Impact of Fake Reviews on Online Visibility: A Vulnerability Assessment of the Hotel Industry. *Information Systems Research*, 27(4):940–961, 2016.
- [LZ16] Luca, Michael; Zervas, Georgios: Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud. *Management Science*, 62(12):3412–3427, 2016.
- [MS22] Muraier, Benjamin; Specht, Günther: DT-grams: Structured Dependency Grammar Stylogmetry for Cross-Language Authorship Attribution. In: *Proceedings of the 32nd GI-Workshop Grundlagen von Datenbanksysteme (GvDB'21)*. CEUR-WS.org, Aachen, 2022.
- [Mu08] Murugesan, San: Harnessing green IT: Principles and practices. *IT professional*, 10(1):24–33, 2008.
- [Ot11] Ott, Myle; Choi, Yejin; Cardie, Claire; Hancock, Jeffrey T.: Finding Deceptive Opinion Spam by Any Stretch of the Imagination. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11, Association for Computational Linguistics, USA, p. 309–319, 2011.
- [Qi20] Qi, Peng; Zhang, Yuhao; Zhang, Yuhui; Bolton, Jason; Manning, Christopher D.: Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020.
- [Ra19] Radford, Alec; Wu, Jeffrey; Child, Rewon; Luan, David; Amodei, Dario; Sutskever, Ilya: Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9, 2019.

- [Sa19] Sanh, Victor; Debut, Lysandre; Chaumond, Julien; Wolf, Thomas: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108, 2019.
- [Sa22] Salminen, Joni; Kandpal, Chandrashekhar; Kamel, Ahmed Mohamed; gyo Jung, Soon; Jansen, Bernard J.: Creating and Detecting Fake Reviews of Online Products. *Journal of Retailing and Consumer Services*, 64:102771, 2022.
- [Sh17] Shahid, Usman; Farooqi, Shehroze; Ahmad, Raza; Shafiq, Zubair; Srinivasan, Padmini; Zaffar, Fareed: Accurate Detection of Automatically Spun Content via Stylometric Analysis. In: 2017 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 425–434, 2017.
- [St21] Strøm, Eivind: Multi-label Style Change Detection by Solving a Binary Classification Problem—Notebook for PAN at CLEF 2021. In (Faggioli, Guglielmo; Ferro, Nicola; Joly, Alexis; Maistro, Maria; Piroi, Florina, eds): CLEF 2021 Labs and Workshops, Notebook Papers. CEUR-WS.org, Aachen, pp. 2146–2157, 9 2021.
- [SW65] Shaphiro, S; Wilk, MJB: An analysis of variance test for normality. *Biometrika*, 52(3):591–611, 1965.
- [TMS19] Tschuggnall, Michael; Murauer, Benjamin; Specht, Günther: Reduce & Attribute: Two-Step Authorship Attribution for Large-Scale Problems. In: Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL). Association for Computational Linguistics, Hong Kong, China, pp. 951–960, November 2019.
- [Zl18] Zlatkova, Dimitrina; Kopev, Daniel; Mitov, Kristiyan; Atanasov, Atanas; Hardalov, Momchil; Koychev, Ivan; Nakov, Preslav: An Ensemble-Rich Multi-Aspect Approach for Robust Style Change Detection. In (Cappellato, Linda; Ferro, Nicola; Nie, Jian-Yun; Soulier, Laure, eds): Working Notes of CLEF 2018 - Conference and Labs of the Evaluation Forum. CEUR-WS.org, Aachen, 9 2018.

## Session 3



# Seamless Integration of Parquet Files into Data Processing

Alice Rey,<sup>1</sup> Michael Freitag,<sup>1</sup> Thomas Neumann<sup>1</sup>

**Abstract:** Relational database systems are still the most powerful tool for data analysis. However, the steps necessary to bring existing data into the database make them unattractive for data exploration, especially when the data is stored in data lakes where users often use Parquet files, a binary column-oriented file format.

This paper presents a fast Parquet framework that tackles these problems without costly ETL steps. We incrementally collect information during query execution. We create statistics that enhance future queries. In addition, we split the file into chunks for which we store the data ranges. We call these *synopses*. They allow us to skip entire sections in future queries.

We show that these techniques only add a minor overhead to the first query and are of benefit for future requests. Our evaluation demonstrates that our implementation can achieve comparable results to database relations and that we can outperform existing systems by up to an order of magnitude.

## 1 Introduction

Data is stored less and less in relational database management systems (RDBMS) [AAS13; A112; Id11; Ka14; OI17]. Instead, users tend to store large amounts of data in data lakes with standardized file formats such as Parquet [ASF13]. Nevertheless, the users still expect the same performance as if the data resides in an RDBMS, which improvised tooling cannot achieve. Even though database systems are much better suited for data exploration, existing RDBMSs that support directly querying files still cannot reach the performance of database relations. They face the problem of not having any insights about the underlying data, which are crucial for efficient data access and query plan optimizations. Usually, an RDBMS knows the processed data well since it can collect all sorts of information while the user loads the data into the database.

Parquet files are one of the most used data structures for storing large amounts of data. Big companies like Twitter [De13], Netflix [WG17] or Skyscanner [SE19] use it to store large amounts of data for big data analysis. The column-wise storage format is close to how an RDBMS with a columnar storage engine would store the data, making the Parquet format a great candidate for integrating it into a data processing pipeline.

The Parquet file format tries to balance data compression and data access. The format supports different encoding and compression schemes. Therefore, additional decoding and

---

<sup>1</sup> Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany, {rey, freitagm, neumann}@in.tum.de

decompression steps are required to access the underlying values, which database relations typically do not have. Furthermore, the Parquet file offers a lot of optional fields such as the minimum and maximum values or the number of null values, which can be helpful for query execution. However, since these fields are optional, we cannot rely on them.

Naive file access would be very costly if the dataset resides on a remote server. The *lineitem* table with 10 million rows of the TPC-H dataset at scale factor 10 has a total size of 2 GB when we store it in a Parquet file generated by Spark [ASF14]. When accessed via HTTP, it would require 16 seconds to download the entire file in a typical local network with a bandwidth of 1 Gbit/s. If we only require one column, a smart access logic can minimize the download time to less than 2 seconds. Parquet files allow us to use byte-range requests to only download required columns. In addition, we also keep structural information about the data in a fixed number of synopses. We split the file into equally sized chunks and track the minimum and maximum value of the contained data, which allows us to skip entire chunks in future queries. Since this technique was already used in previous works under different names [E113; La16; Mo98; O117; Zi17], we chose the generic name *synopses*. To keep the computational overhead of these synopses small, we only compute the data ranges for columns that are currently required. If a user requests additional columns in the future, the corresponding ranges can be added incrementally.

Most users will switch to more complex analytical queries after an initial exploratory phase, where a fully-fledged database system with a query optimizer comes in handy. Database systems usually perform great on complex queries since they have prior knowledge about the data gathered while copying it into database relations. Since we skip this step, we save initialization costs and avoid accessing data we would never use. On the other hand, we lose crucial information that might have helped during future query optimization steps. Therefore, we take the Parquet view feature one step further by allowing a smooth transition from exploratory data analysis to more complex analytical queries. We will not add an initial scan over the Parquet file, but instead, we will collect samples and sketches of the used data while executing the queries. We do not expect complex operators during the exploratory phase so that we can accept some slowdown. We then benefit from the information we gathered during the exploratory phase for later queries. To the best of our knowledge, our implementation is the first to reach comparable execution times to database relations for Parquet files with these techniques.

The key contributions of this paper are:

1. A smart access logic for Parquet files that introduces minimal initial overhead to the query execution time and improves performance over time.
2. An incremental procedure for cheap statistics computation that enhances query optimization steps for future queries.
3. A remote access strategy that minimizes execution time overhead.

The remainder of this work is structured as follows: We start by briefly explaining the challenges the Parquet file format introduces to an RDBMS in Sect. 2. Second, in Sect. 3, we explain the techniques we used to conquer the imposed challenges and how scanning Parquet files can be integrated into database systems. Third, we perform a detailed evaluation of our implementation in Umbra [NF20] with different benchmarks (Sect. 4). Finally, we conclude with some related work in Sect. 5 and summarize the work in Sect. 6.

## 2 Background

Parquet files are an excellent match for databases that use a column-wise storage format due to their columnar file format. Nevertheless, Parquet files pose some challenges when integrating them into query processing. In this section, we will discuss the structure and versatility of the Parquet format in the first two subsections to understand the need for certain design decisions. Finally, we conclude the section with the challenges that the file format introduces.

### 2.1 Parquet File Format

Parquet stores data in columnar format and is not human-readable in contrast to CSV or JSON files. Instead, the goal of the Parquet format is to store data as densely as possible. Fig. 1 (b) shows the basic structure of a Parquet file. The data is first separated into row groups that split the dataset horizontally. Each row group is then split into column chunks vertically, storing each column in a separate column chunk. The actual elements of the column chunk are stored on one or more data pages. Parquet does not enforce the number of rows which should be stored per row group or page. In addition, the number of rows stored per page does not need to be synchronized between the column chunks of the same row group or the same column. In our example, we store the values of  $x$  on two pages in the first row group and on three pages in the second row group. Column  $y$  stores the values on four pages and column  $z$  on one page for both row groups.

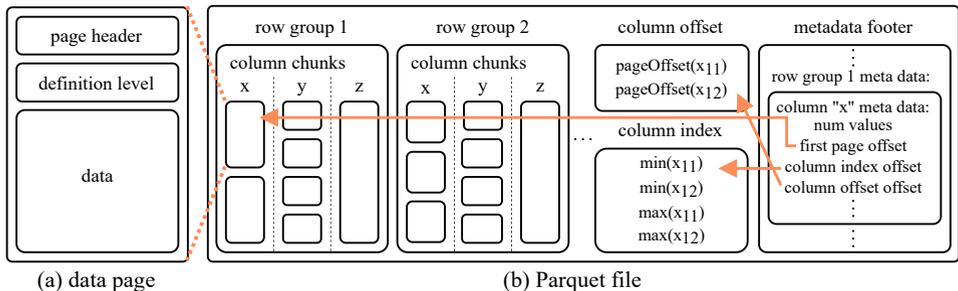


Fig. 1: Parquet File Structure

The schema and data arrangement are stored in the Parquet file's footer. In our example, we display the first row group's metadata of column  $x$ . In addition to much other information, the column chunk has to store the offset of the first page as the starting point. The *column index* and *column offset* data structures, which are usually located close to the metadata footer, are optional and can be referenced via their offset. The *column offset* stores the offsets to all pages of the column chunk, and the *column index* stores the minimum and maximum value of each page of the column chunk.

In Fig. 1 (a), we display the general structure of a data page. Each page starts with a page header that contains the page type, the number of contained values, the used encoding, and the compression. The page writers we examined used SNAPPY [GG11] as the default compression format for pages. The most commonly used page encoding we found is dictionary encoding, where the actual values are stored on a dictionary page, and the data pages store indices into the dictionary. The number of bits required to store the dictionary indices depends on the number of elements in the dictionary. The indices are stored in multiple run-length encoded or bit-packed encode runs. Nevertheless, the encoding can switch to plain pages if the number of distinct elements exceeds the dictionary size.

Parquet also supports nested data types with the record shredding and assembly algorithm presented in the Dremel paper [Me10]. Datasets containing nested types violate the first normal form, which poses a fundamental challenge for any relational database management system regardless of the specific data storage format [DLN21; Sh99]. Our framework is able to scan any nesting level. Connecting multiple levels is a problem we consider to be orthogonal to the objective of this paper. We plan to investigate possible solutions in future work using our current framework as the base layer. The main idea of the Dremel algorithm is to use definition and repetition levels to store how many elements belong to the same parent element and at which level a value is undefined. The *definition level* is located below the header and is also relevant for nullable columns. If the column chunk is nullable, each page keeps a *definition level buffer* at the beginning of the page, which stores for each row if it is NULL with run-length and bit-packed-encoded. The rest of the page contains the actual values. Since the definition level already encodes NULL values, only non-NULL values are stored in this section. Parquet supports a fixed set of physical types such as *BOOLEAN*, *INT64*, or *BYTE\_ARRAY*. All other types combine a physical type with a converted type. To store decimal values in a Parquet file, the physical type *INT64* is combined with the converted type *DECIMAL*. The optional fields *precision* and *scale* store additional information about the type.

## 2.2 Versatility of Parquet Writers

In the previous section, we described the general structure of a Parquet file. However, the data in a Parquet file can be spread over the row groups and the pages using any encoding and compression the writer or user wants. This freedom leads to inhomogeneous files even though they have the same file format. Our goal is to have a fast Parquet framework that

can achieve good performance independently of the used Parquet writer. Therefore, in this section, we look at three different Parquet writers to show how much Parquet files differ even though they store the same data.

| Generator    | Rows per Row Group | Pages per Row Group | File Sizes |        |       |
|--------------|--------------------|---------------------|------------|--------|-------|
|              |                    |                     | SF1        | SF10   | SF100 |
| CSV          | -                  | -                   | 719 MB     | 7.2 GB | 74 GB |
| Spark        | 3,000,000          | 150                 | 192 MB     | 2.1 GB | 20 GB |
| uncompressed | 3,000,000          | 150                 | 333 MB     | 3.3 GB | 33 GB |
| DuckDB       | 100,352            | 1                   | 281 MB     | 2.8 GB | 28 GB |
| Arrow        | 67,108,864         | 15 - 1800           | 189 MB     | 2.0 GB | 20 GB |

Tab. 1: Parquet Writer Comparison

In Tab. 1, we listed the properties of three different writers: Spark [ASF14], Arrow [ASF16], and DuckDB [RM19]. To measure their differences, we let each generator write the *lineitem* relation of the TPC-H benchmark to a Parquet file for the scale factors 1, 10, and 100. Apart from the size of the generated Parquet files, we listed the size of the underlying CSV file created by the TPC-H generator to show the compression ratios that the different writers achieve. The *lineitem* relation contains 6 million rows times the respective scale factor. We also included a version where we force Spark not to use any compression to measure the benefit of page-level compression with SNAPPY.

For each generator, we measure the number of rows and the number of pages that are stored per row group. The Spark and DuckDB Parquet writers store a fixed number of elements per page and a fixed number of pages per row group. Since Parquet does not force synchronization between the column chunks, there are writers such as Arrow that do not store the same number of elements per page. Arrow uses a fixed data page size between roughly 0.5MB and 1 MB. For DuckDB and Spark, the page sizes vary from 0.5 MB to 6 MB. Out of all three writers, DuckDB has the worst compression ratio of 2.6. The major difference between DuckDB and the other two formats is that DuckDB does not use any encoding for the columns of the *lineitem* table. Spark and Arrow have a very similar compression ratio of 3.6. If we force Spark to write the pages uncompressed into the Parquet file, the resulting file size is less than 20 % bigger than the DuckDB file. Even though we only cover three different Parquet writers, we have already observed two extremes. DuckDB and Arrow do not take advantage of the hierarchical data layout: DuckDB will only use one page per row group, and Arrow stores the entire dataset in one row group for scale factor 1 and 10 since each row group stores 67 million rows.

### 2.3 Parquet Format Challenges

In the two preceding sections, we pointed out the structure of the Parquet files and their versatility. We will conclude our preliminary discussion with the format’s challenges on

the Parquet integration. We highlighted the freedom Parquet writers have in Tab. 1. The consequence of this freedom is that we cannot assume anything about how the data is split across the Parquet file hierarchy. Unfortunately, especially for parallel processing, this freedom makes scanning more difficult. If Parquet, for example, forces all writers to store the same amount of elements per page for each column, this would allow more optimized accesses. The elements of one row would all be stored at the same page index and the same page offset in the different column chunks, at least for non-nullable columns.

Parquet offers a lot of data structures such as the *column index* that greatly facilitate data access since we can directly jump to the page that contains the required value. Nevertheless, a lot of these structures are optional. If the data structure does not exist, we need to access all preceding page headers to compute the offset of the required page. Parquet’s encoding and compression techniques can minimize the size of Parquet files. On the other hand, it adds additional decompression and decoding steps when reading the data. Dictionary encoded pages use multiple run-length or bitwise encoded runs. Each run stores the number of values it encodes in its header. If we search for an element at a specific page offset, we have to access each run to compute the start of the following run until we reach the required offset.

The definition level encoding is another limiting factor. Storing only non-NULL values on the page helps decrease the overall size of the file. However, if we want to access an element at a specific page offset, we have to scan the definition level first to count the number of preceding non-NULL values to get to the actual offset. Some data types stored in Parquet also introduce additional transformation steps. For example, *BYTE\_ARRAY* elements are represented as 32 bit values for the length followed by the actual value. We transform these into fixed-length 16 byte fields in our internal buffer. If the strings are longer than 12 bytes, we store the size of the string and the first 4 bytes of the value in the upper 8 bytes. The lower 8 bytes contain a pointer that points to the location where the entire string is stored [NF20].

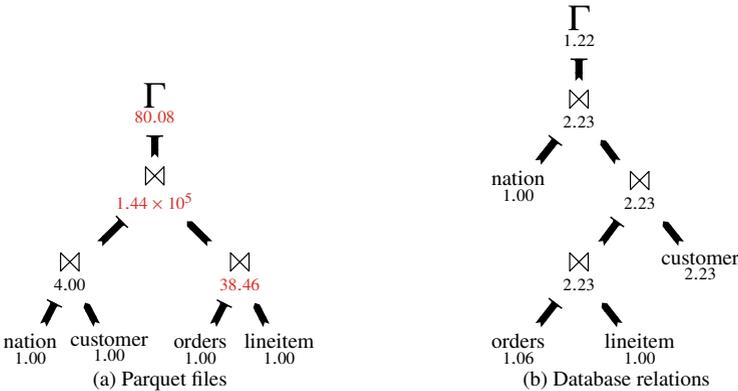


Fig. 2: Q-errors of the cardinality estimates of TPC-H query 10 with scale factor 10.

The only statistics available in Parquet files are the cardinality of the contained dataset and each page’s minimum and maximum values. Unfortunately, the minimum and maximum

values are optional fields, so Parquet writers are not forced to use them. Out of three Parquet writers we analyzed, only Spark stores minimum and maximum values. These minimum and maximum values, as well as the cardinality of the datasets, are the only sources available for performing cardinality estimates. Therefore, we get imprecise results since we do not know how the data is distributed within the given boundaries. As a consequence, we get erroneous cardinality estimates and suboptimal query plans.

For efficient query processing, it is necessary to produce optimal query plans independently of the amount of metadata the Parquet file provides. Since cost-based query optimizers heavily rely on statistics and samples gathered during table initialization, the optimizer suffers a lot when these do not exist. In Fig. 2, we visualize the optimized query plans of TPC-H query 10 based on Parquet files on the left and database relations on the right. For each operator, the optimizer tries to estimate the resulting cardinalities. The q-error next to each operator gives us the deviation of the estimates from the actual cardinality [MNS09]. In the case of the Parquet file version, the estimated result cardinality of the join between *orders* and *lineitem* is off by a factor of 38. The q-error gets even worse with the following join with a factor of 144 thousand, which is the reason for the suboptimal query plan for the Parquet files. This shows how crucial a good cardinality estimate is for a Parquet scan to be an acceptable alternative to database relations. The Parquet scan cannot get close to the execution times of database relations as long as the query optimizer cannot choose the same query plans for the Parquet files.

### 3 Integrating Parquet Files into Query Processing

We now explain the different techniques for integrating Parquet files seamlessly into a query processing pipeline. We tackle their complex structure, versatility, and lack of metadata described in the previous section. We propose a parallelization technique with stable performance, independently of how the data is distributed over the Parquet file hierarchy. In addition, we minimize the amount of data we have to access, which is especially beneficial for remote files. Thirdly, we present a technique for collecting information about the dataset that benefits future queries without notably sacrificing the execution time of the currently executed query.

Figure 3 describes the concept of how future queries can benefit from information gathered in preceding scans. We visualize a simplified execution of three different queries, executed sequentially from left to right. All three queries access the same Parquet file during the *Parquet Scan* phase. The Parquet file is split into four row groups ( $r1$ ,  $r2$ ,  $r3$ ,  $r4$ ) and contains three columns, namely  $x$ ,  $y$ , and  $z$ . Throughout the three queries, we incrementally collect information about the Parquet file. In the *statistics* we store HyperLogLog [F107] sketches to estimate the distinct values. We also keep a small set of rows as a sample for multi-column estimates and predicate selectivity estimates [FN19]. The *synopses* store the minimum and maximum value of each column chunk for each row group. To limit the required space of synopses, the user can choose an upper limit for synopses. If we have more row groups

than synopses, multiple row groups are grouped into one synopsis. With the help of the statistics, further queries can choose better query plans during the query optimization phase. The synopses can be used in future queries to skip the row groups that do not meet the restrictions of the queries. This is only applicable if the data in the Parquet file is implicitly clustered, which is often the case in real-world datasets [Mo98; Vo18].

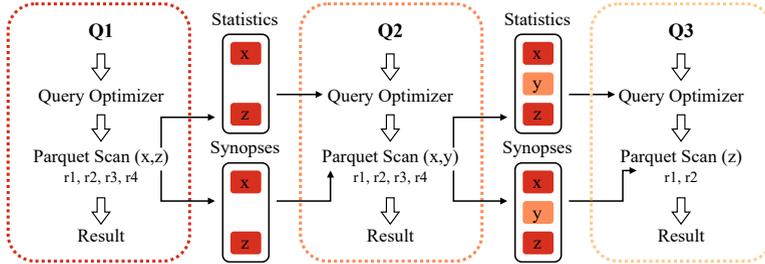


Fig. 3: Sequential execution of three queries (Q1, Q2, Q3) with the incremental computation of synopses and statistics for the accessed Parquet file.

The first query (Q1) in Figure 3 requires the columns  $x$  and  $z$ . We do not have any prior knowledge about the Parquet file during the query optimization phase, which we could consider. Therefore, we most likely end up with a suboptimal query plan. During the Parquet scan, we will access the column chunks of  $x$  and  $z$  of all row groups. We store statistics about those two columns and leave space between them for the second column  $y$ . We also track the data ranges of all four row groups for both requested columns and store them in the synopses. The next query (Q2) requests the columns  $x$  and  $y$ . The query optimizer can consider the statistics for column  $x$  that we gathered during the first query, which will help select a better query plan. While scanning the Parquet file for the second query, we will gather information about the column  $y$ , which we can use to fill in the gap in the statistics and the synopses.

Since Q1 and Q2 cover all columns, any following query will not have to compute any statistics or synopses. For query 3 that requests column  $z$ , the query optimizer can work with the statistics and develop a cost-optimized query plan as it could for any database relation. During the *Parquet Scan* phase, we can take advantage of the synopses. Query 3 restricts column  $z$ . Since we know the minimum and maximum values, we can check if the requested range and the row group range overlap for each row group. In the case of the third and fourth row groups, the ranges do not overlap. For this reason, we will only process the first and second row groups, which we reference with  $r1$  and  $r2$  in Figure 3. Query 2 cannot take advantage of the synopses yet, since we still need to gather information about column chunk  $y$  while executing that query. To compute correct statistics for column  $y$ , we need to consider all row groups of that column.

To achieve stable parallelization over the Parquet file, we looked at different possible ways of splitting the Parquet file, which we visualized in Fig. 4. We split the entire workload into

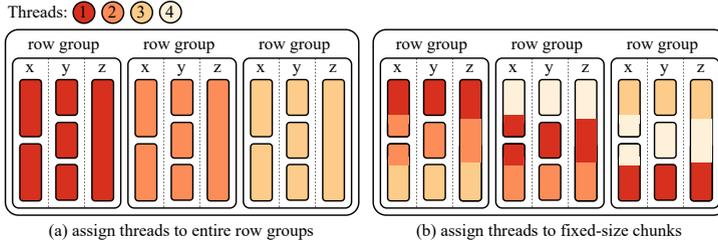


Fig. 4: Comparison of different parallelization approaches for Parquet files.

smaller work units to be able to process them independently of each other by all threads in a work-stealing framework such as the one provided by Umbra [NF20]. In our example, the Parquet file consists of three row groups that contain 60,000 rows each. We assume that we have four threads available for this Parquet scan. The sections that each thread is scanning are colored in the corresponding color.

In Fig. 4 (a), each thread is assigned to an entire row group. One of the threads is idle in this case because we do not have enough row groups in the Parquet file. As we know from Tab. 1, there are Parquet file generators such as Arrow that only use one row group to store 60 million rows. In that case, all threads except for one would be idle. Therefore, we opted for another more adaptive approach, as visualized in (b). We work with a fixed-sized chunk size. For this example, we assume a chunk size of 20,000 rows. Our goal is to keep all threads busy while ensuring that the helper structures required for the scan do not take up too much space. Therefore, we limit the number of row groups we process in parallel. We tested different spaces between 265 MB and 4 GB and achieved the best results by limiting the required space to 1 GB.

### 3.1 Parallelizing the Parquet File Scan

Achieving stable parallelization over Parquet files requires parallelizing the scanning below row group level. We split the Parquet files into independently processable chunks to reach similar performance to queries that are based on database relations and executed in a work-stealing framework. The maximum granularity level is the row group size. We further split each row group into fixed-sized chunks, which we call morsels [Le14], depending on the row group size. The threads that are processing morsels of the same row group will have to access and prepare the same file ranges. We therefore reduced the amount of duplicate work as much as possible while avoiding contention. In Fig. 5 we describe how we parallelized our framework and how the different aspects we already mentioned work together.

During the setup of the scan, we use the synopses to filter out row groups that do not meet the restrictions, which are the filter predicates of the query (Line 3). If the synopses are

```

1 def setupScan:
2   if !computeStatistics:
3     requiredRowGroups = synopses.filter(rowGroups)
4   else:
5     if noStatsAvail: computeSamplePos()
6     else: loadSamplePos()
7   computeConcurrentRowGroupCount()
8   rowSize = sum(requiredColumnSizes)
9   blockSize = bufferSize / rowSize
10  bufferPointers = computeColumnWriterOffsets()
11
12 def prepareDictionary(column c, dictionary d):
13   if d.isCompressed:
14     singleAccessBlocking() => {
15       rowGroupBuffer[c][d] = decompress(d)}
16   if c.type == 'BYTE_ARRAY':
17     singleAccessBlocking(convertDictionaryEntries)
18
19 def prepareDataPage(column c, page p):
20   if p.isCompressed:
21     singleAccessBlocking() => {
22       rowGroupBuffer[c][p] = decompress(p)}
23   if p.dictEncoded:
24     prepareDictionary(c, rowGroupBuffer.dictionary)
25   if p.dictEncoded or c.type == 'BYTE_ARRAY':
26     singleAccessNonBlocking(computeSparseOffsets)
27
28 def moveToOffset(row, column):
29   page = moveToPage(row)
30   prepareDataPage(column, page)
31
32   if sparseOffsetsComputed:
33     moveToSparseOffset(row)
34     moveToRowOffset(row)
35
36 def readColumn():
37   moveToOffset(morsel.currentRow, column)
38   insertElements(column)
39
40 def readNextBlock(morsel):
41   if computeStatistics:
42     for column in reqColumns: readColumn()
43     computeStatistics()
44   else:
45     matches = list(range(0, blockSize))
46     moveToPageWithMatches(restrictedColumns)
47     for column in resColumns:
48       readColumn()
49       checkRestrictions(matches)
50     for column in unresColumns: readColumn()
51
52 setupScan()
53 threads.doInParallel((morsel) => {
54   # wait until rowGroup in rowGroupBuffer
55   singleAccessBlocking(setupRowGroupProps)
56   while not morsel.allRowsProcessed():
57     readNextBlock(morsel)
58     processBlock()
59 })

```

Fig. 5: Pseudocode for scanning Parquet files in parallel

not available yet, or if we need to compute statistics for a column that is accessed for the first time, all row groups will be processed. As part of the statistics, we also retrieve a data sample. If the file is accessed for the first time, we compute 1024 random sample positions (Line 5). We allocate enough space to store the sample values of all columns. Then, when the user queries the file again and requires other columns this time, we incrementally add samples for these columns as well. We also choose the number of row groups we will process in parallel such that all threads are busy while the required space is limited. The next step is to prepare for each thread a fixed-sized buffer in which we will write the values of all required columns (Lines 8 to 10). At first, we compute the space we would require to store one row, which is the sum of the sizes of all required columns. We group the data inside the buffer by column for the vectorized evaluation of restrictions and to be able to copy entire blocks if possible. We use a fixed-sized buffer between the Parquet file layer and the next operator since it allows consistent processing of all columns independent of how they were stored inside the file. For strings, the database engine has to use an out-of-line format to allow fixed-sized elements. Many columns are dictionary encoded by the writers we examined in Tab. 1 and many data types require additional transformation steps to data types available in the RDBMS. Therefore, we have to touch most of the tuples either way and storing them into a buffer only adds minor overhead. The buffer makes our approach applicable for both push-based execution models [Ne11] as well as vectorized execution models [BZN05].

After the initial setup, all threads can work in parallel (Line 52). The row groups are split into morsels that are assigned to free threads until no morsels are left. Whenever a thread starts processing a specific morsel, the thread has to check at first if the currently required row group is loaded into the row group buffer. Then, it sets up the data structures for the row group which is only done by one morsel per row group. Afterwards, the thread processes the morsel in `blockSize` chunks that fit into the fixed-sized buffer by repeatedly calling the `readNextBlock` function until no data is left.

For each column, we will at first move to the correct offset and then insert the elements at the correct position in the buffer, given by the `bufferPointers`. Each morsel can independently compute the start position of the current block since we know how many rows are stored per row group and how many values are stored per page. Given the currently required row number, we will move to the page that contains the row we need, prepare the page and then move to the correct offset inside the page. For the page preparation (Line 19), we let only one thread decompress the page and store it in a buffer where all other morsels can access it afterwards. If the page is dictionary encoded (Line 12), one morsel per row group decompresses the dictionary and converts the dictionary into a fixed-sized out-of-line representation in case of a *BYTE-ARRAY* column. For all other data types, the index can be multiplied by the data type size to get to the correct offset in the dictionary page. The same holds for data pages. If Parquet stores the values plainly, we can get to any offset by multiplying the required offset by the data type size. If the data page is dictionary encoded or stores plain *BYTE-ARRAY* values, we can not easily jump to a certain offset. Given we have multiple morsels that work on the same page, they all have to start reading from the start of the page to reach their required offset. Letting one morsel compute sparse offsets distributed over the data page (Line 26) helps future morsels to reach the required offset earlier. To avoid any contention at this step, we do not let the other threads wait until the offsets are available. If they are available, the other threads can use them (Line 31). In the last step we move to the actual row offset, either starting at the beginning of the page or starting from the closest sparse offset if available.

For the loading step (Line 37), we favor copying whole data blocks at once for each column, which is only possible for fixed-size, plainly stored, non-nullable columns that need no transformation step. For nullable columns, we first have to scan the run-length-encoded and bit-packed-encoded runs of the *definition level buffer*. Since the pages store non-null values densely, we will have to check how many non-null values we have to read. In our buffer, we do not store the non-null values densely. Instead, we store the null information for each tuple in one byte and leave an empty gap if the element is null. Leaving gaps for null values makes the access cheaper since we can precompute all offsets independently of other rows.

If we access columns for the first time, we will compute statistics and synopses for each block independently after all the values of all columns are loaded into the buffer (Line 42). If the Parquet file stores min/max information, we use these to compute the synopses only once per row group.

If we do not have to compute statistics (Lines 43 to 49), we will at first skip as many pages as possible based on our restrictions and the min/max information from the Parquet file. Afterwards, we start loading the restricted columns one after the other into our buffer and evaluate the restrictions with vectorized functions. In the `matches` list, we store which indices of our current block are still valid. At the beginning, all indices are stored in the list since all are valid. Each restricted column evaluates its restriction on its values and updates the `matches` list accordingly. Further restricted and unrestricted columns use the `matches` list to only load the values of rows that are still valid.

### 3.2 Parquet File Access

The data retrieval becomes our main bottleneck if the Parquet file is located on a remote server. We will not download the entire file at once since this would include a lot of unused data, especially if the user only requires a small fraction of the entire dataset. We can guarantee that we only request the required data with byte-range requests, given our remote source is accessible via HTTP. As with query 3 in Fig. 3, we can use the synopses computed in preceding requests to exclude row groups that do not match the restrictions of the query, and we only retrieve the column chunks that the query requires.

During data exploration, users will likely access the data of a queried Parquet file again in the following queries. Therefore, if the user works on a remote Parquet file, it is crucial to cache the used data locally to reduce the access costs for future queries. Since we want to integrate the Parquet files into an existing database system, we can utilize its buffer manager. The buffer manager in Umbra is based on LeanStore [Le18] with the addition of variable-size pages [NF20]. At first, we will request the size of the Parquet file with a `HEAD` request. If the file is smaller than 64 KB, we will download the entire file and load it into one 64 KB page. Otherwise, we will first access the metadata block at the end of the file to get some general information about the contained data, such as the data types, the number of row groups, and the cardinality. Then, when our morsels start processing, we will request the data as needed. Each column chunk is requested separately and stored on a buffer page. The only difference between the Parquet file pages and the standard database pages is that we drop the page instead of writing it to disk when it is evicted. Therefore, we can not reload an evicted page, but instead we have to download it from the remote source again.

For local Parquet files, we could, in theory, work with memory-mapped files, but its usage was discouraged by Crotty et al. [CLP22]. Therefore we treat local and remote Parquet files the same and load both as needed into our buffer manager.

### 3.3 Query Plan Optimizations

Databases heavily rely on cardinality estimates when it comes to query plan optimizations. This information is usually provided by the user or computed by the database engine. The

user can give hints to the database with primary and foreign keys. The database computes its metadata during the *INSERT* statement. We compute HyperLogLog sketches for single columns and retrieve samples from our data [FN19; NF20].

When we access our Parquet file for the first time, we do not have any prior knowledge about the contained data. Nevertheless, we do not want to waste time with an additional initialization step before starting to work on the actual query. Therefore, we will collect the information we need while executing the first query. We can use the statistics for query plan optimizations starting from the second query. Typically, users will start with exploratory queries when they access Parquet files for the first time. These queries are simple and not very complex, so we do not need metadata to get a relatively good query plan estimate. However, suppose the users still want to execute complex queries on a Parquet file immediately. In that case, they could trigger the computation of the statistics beforehand by accessing all required columns with a simple scan request. For the first execution, we use vague estimates for the query plan optimization: For each column chunk that uses dictionary encoding, we use the number of values in the dictionary as a distinct value estimate. This estimate can be wrong since the encoding of a column chunk can change back to plain encoding.

During execution of the first query, we will compute HyperLogLog sketches and store a data sample from the file for each required column, similar to the synopsis computation. For the sample, we compute 1024 random row numbers (Line 5 in Fig. 5). We allocate enough space to store the sample values of all columns. Then, when the user queries the file again and requires other columns this time, we will incrementally add sketches and samples for these columns as well. Since strings are stored using a fixed-sized, out-of-line format, and all other types required to store the Parquet types have a fixed length, we can precompute the required space we need. All morsels can fill the sample in parallel since the sample positions are stored in a sorted list. Each morsel knows its row range in the Parquet file and can independently check if it contains one or more of the sample positions. The index in the sample positions list tells the thread the offset of the row in the sample. We can compute the required position in the data sample with the sample positions' index, the index of the required column in the schema, and the type size. Since strings are stored using an out-of-line string format, we allocate additional space at the end of the data sample for string columns, as needed. The string format has to work with offsets to allow relocations of the data sample when the data sample outgrows its current memory location.

In Fig. 5, we depict how the morsels are scanned: We fill our fixed-sized buffers with as many rows as possible and then push them to the parent operator. Before we call the parent operator, we reuse the buffer representation to cheaply retrieve all the information we need for our sketches and statistics computation (Line 42). This step saves much time since we do not have to decode and decompress the values from the Parquet file separately. We mentioned that we first load the restricted columns into the buffer and check the restrictions before loading the values of the qualifying rows for the remaining columns. To compute correct statistics, we need all rows, and we will therefore load all values when statistics have to be computed, even though they do not meet the restrictions.

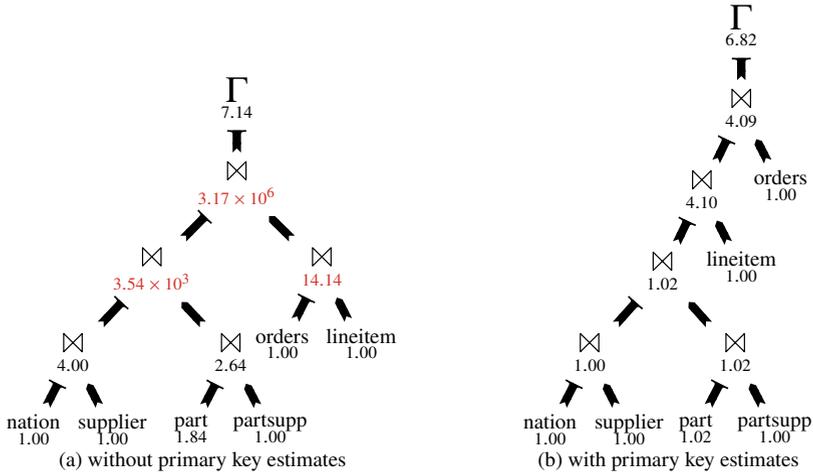


Fig. 6: Q-errors of TPC-H query 9 with scale factor 10 on Parquet files.

The data sample and the HyperLogLog sketches already help to generate optimal query plans. Nevertheless, we still have a significant performance loss for queries that heavily rely on primary key information, which we do not have in Parquet files. For example, query 9 of the TPC-H benchmark contains multiple primary/foreign key joins. The impact of the primary key information can be seen in Fig. 6. The query plan without primary key information is displayed in Fig. 6(a). Fig. 6(b) displays the usage of primary key estimates. Without primary key estimates, the q-errors are significantly higher, which leads to a suboptimal query plan.

To enhance query plans such as the one depicted in Fig. 6, we try to retrieve the primary key from the given Parquet file. A lot of work has already been done on exact methods for the detection of unique column combinations (UCCs) and functional dependencies (FDs) [AGN15]. However, discovering all minimal UCCs is an NP-hard problem. Checking if a column combination is unique, is very costly, therefore we only work with estimates here. In the literature, some approaches already exist that define different heuristics either to select a primary key from a given set of primary key candidates [JN20; PN17] or to estimate them from scratch [MK17].

Since we know the query already, we can also benefit from the query plan properties, as already discussed by Andersson [An94]. We need the primary keys in our example to know whether a join is a primary key/foreign key join. Therefore, we only consider those columns used in an equality condition of a join operator. We might miss the correct primary key. However, if the primary key is not part of the join condition, knowing the primary key would be useless since it does not influence the join order. For our approach, we only look at possible primary keys that consist of one or two columns to keep the complexity of our

estimates low. In addition, according to Papenbrock, Naumann, and Jiang [JN20; PN17], it is more likely to have short keys because it is easier to work with these.

In their work [MK17], Motl and Kordík present different features that they consider for their primary key predictions. According to their evaluation, the ordinal position is the most relevant factor. Therefore, we check the columns in positional order. The first column used as an equality condition by a join operator that has a distinct values estimate close to the cardinality of the relation will be our estimated primary key. If we cannot find a single-column candidate, we repeat our search from the beginning with column pairs. Both columns have to be a part of an equality comparison in a join, and their combined selectivity should select almost only one tuple to be considered the primary key. We can compute very accurate distinct value estimates for single columns with the HyperLogLog sketches we computed during the first query execution. Our combined selectivity estimates are based on our retrieved samples.

## 4 Evaluation

In this section, we measure the performance of our Parquet scan framework, which we implemented in Umbra. We start with the statistics and synopses computation. We evaluate the computational overhead they introduce and their benefit by measuring the speedup the statistics introduce and the data transfer savings of the synopses. The second step is a look at the scalability of our system. Afterward, we compare our system with three other systems, namely DuckDB [RM19], Hyper [KN11], and Trino [Se19; TSF20]. We end our evaluation by comparing the performance of Parquet files and database relations. We ran all our experiments on an Intel Xeon Gold 6338 CPU with 32 physical cores and 64 logical cores running at 2.0 GHz. The server has 256 GiB of main memory, and all Parquet files were placed on a local Samsung 850 Pro SSD with 2 TB of storage space. For our benchmarks, we used the decision support benchmark TPC-H with different scale factors and the join order benchmark (JOB) [Le15]. All measurements were repeated ten times, and we always selected the fastest execution to measure performance with warm caches.

### 4.1 Impact of Statistics and Synopses

When a user accesses a Parquet file for the first time, we will compute statistics and synopses. This section evaluates the overhead and the benefit of the statistics and synopses for future queries using the TPC-H benchmark with scale factor 10. In Fig. 7, we display the execution time of the first and second execution for all 22 TPC-H queries. By *first execution*, we mean a pass where we compute statistics while running the query. For the *second execution*, we have statistics available that we can use to optimize our query plan. The *first execution* and the *second execution* are repeated 10 times. For the *first execution*, we distinguish between the statistics computation and the query execution time. On average, the statistics

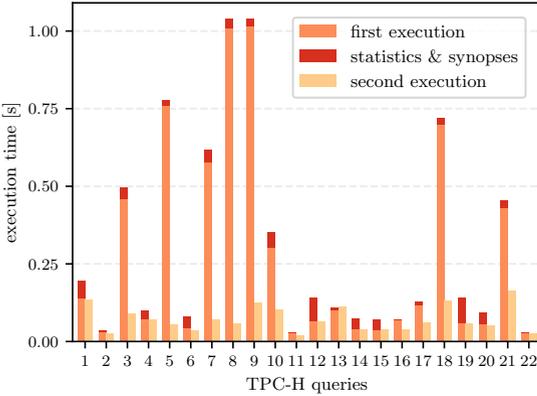


Fig. 7: Overhead and benefit of computing statistics on the performance of Parquet scans in Umbra.

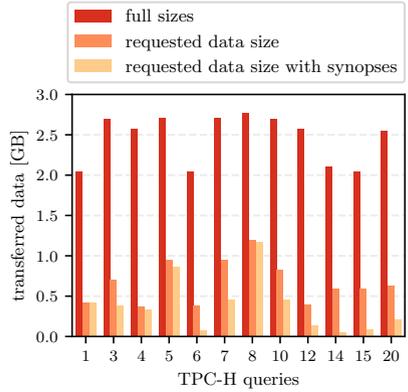


Fig. 8: Impact of synopses on the amount of transferred data for the TPC-H queries.

and synopses computation takes up 20% of the overall execution time of the first pass. In addition, the statistics and synopses for the *lineitem* Parquet file of the TPC-H dataset only require 260 KB of space, independently of the scale factor when we limit the number of synopses to 100. Even though there are queries that do not benefit from the statistics computation, since the naively chosen query plan is already optimal, they speedup the queries by 3.6 on average.

To demonstrate the benefit of the synopses, we assume the TPC-H datasets are sorted by the timestamp columns. In real-life datasets, we will most likely have some bias [Mo98; Vo18]. For example, the data might have been collected over time, which results in sorted timestamps. We used the compressed Parquet dataset with a scale factor of 10 for the measurements and stored the data on a remote server. In Fig. 8, we visualize the amount of data we have to transfer from the remote server to evaluate the TPC-H queries that are restricted by timestamps. We compare the total size of all required files with the amount of data we request from the server, once without synopses and once with synopses available for each row group. Even without the synopses, the amount of data we request is significantly smaller than the actual file sizes since we only request the required column chunks. On average, we request a fourth of the total file size. The benefit of the synopses is dependent on the restrictiveness of the queries. For queries 1 and 8, we cannot exclude any row groups, but for query 14, only one of the sixteen row groups meets the restrictions.

## 4.2 Scalability

Since we work with the buffer manager of Umbra to process local and remote Parquet files, we can also process files bigger than the main memory. We measured the execution times of all TPC-H queries for scale factors 1, 10, 100, and 1000 and displayed them in Fig. 9 with a

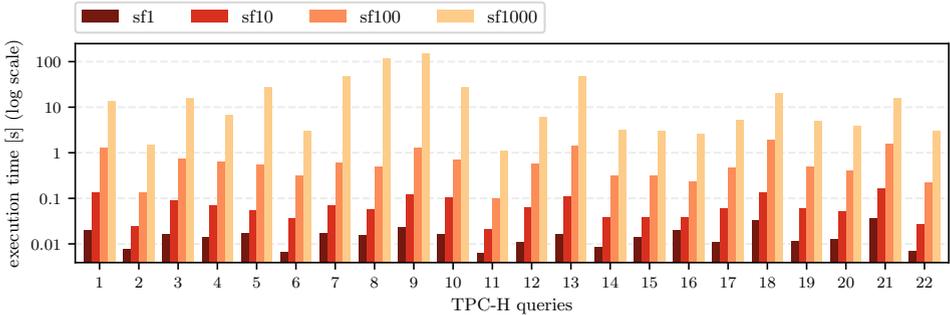


Fig. 9: Execution time of TPC-H queries with different scale factors.

logarithmic scale with base 10. Since the server in our experiments has 256 GiB of main memory, the *lineitem* Parquet files for scale factor 1000 do not fit into the buffer manager of Umbra. All queries are repeated ten times, meaning the data will be preloaded in the buffer manager if possible. The results show that our approach is stable and scales well on growing workloads. There are some queries where scale factor 1000 is more than 10 times slower than scale factor 100, because large chunks of the Parquet files have to be accessed for these queries, which do not fit into our buffer manager at once, so we have to evict pages. We also scanned all columns of the Parquet file *lineitem* and explicitly cleared the cache beforehand for all four scale factors. In this case, we observed perfect scalability.

### 4.3 System Comparison

We will now compare our implementation with the Parquet views of other systems. Apart from the two RDBMSs, DuckDB and Hyper, we also included a distributed query engine, namely Trino. For these measurements, we used the Trino CLI (v402) and the respective Python APIs for DuckDB (v0.4.0) and Hyper (v0.0.15530). To compare the systems, we measure the TPC-H benchmark with a scale factor of 10 and the JOB benchmark for this evaluation. We use the Parquet files generated by Spark, compressed and uncompressed, to evaluate the additional costs of the decompression. Fig. 10 shows the relative speedup of Umbra compared to the other three systems. We grouped the sets by the benchmarks into two separate graphs. We used a logarithmic scale for the speedup factor. We can see that our system outperforms all other systems for all queries. Moreover, our implementation is rarely less than twice as fast as the compared systems. We only display the speedups over the files generated by Spark since the way it distributes data is between the extremes of DuckDB and Arrow, but our system outperforms the other systems for all three writers. The geometric mean speedup for Umbra compared to DuckDB is  $7.5\times$  on the TPC-H benchmark and  $13\times$  for the JOB benchmark for the compressed versions. Compared to Hyper, Umbra is, on average, for both benchmarks  $13\times$  faster. Trino is the slowest of all four systems. For the TPC-H benchmark, we measure a geometric mean slowdown of  $21\times$  for the compressed

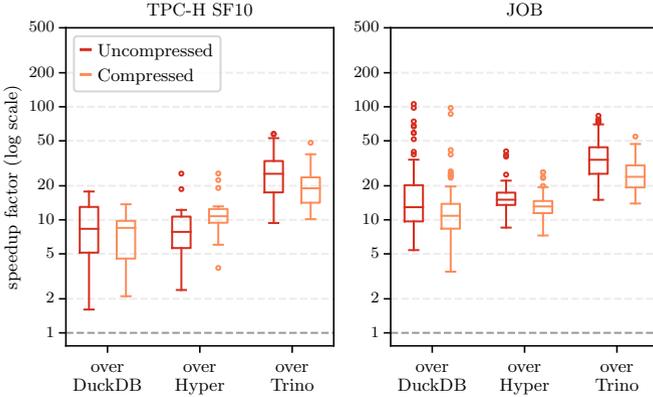


Fig. 10: Speedup of the Parquet scan of Umbra over the Parquet scans of DuckDB, Hyper, and Trino.

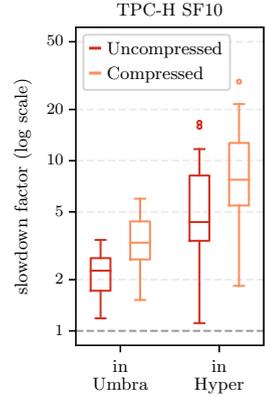


Fig. 11: Slowdown of Parquet files over database relations.

version. For the JOB benchmark, the compressed version is 26× faster in Umbra on average. The speedup for the uncompressed versions is mostly higher than for the compressed versions. Our system is significantly faster than the other systems because of our advanced statistics and sample computations. In addition, our approach allows parallelization beneath the row group level, which makes the approach more stable.

To factor out the impact of the query plan optimization, we reran the TPC-H benchmark for Hyper and Umbra and forced both systems to use the same query plans. To exclude the efficiency of other operators, we will execute the queries once on Parquet files and once on database relations in both database systems. We then compare the slowdown factors of the Parquet scans over their respective database relation versions. Fig. 11 visualizes the slowdown of Parquet files for the TPC-H benchmark for scale factors 1 and 10. We grouped them by the respective benchmark and used a logarithmic scale. We display the uncompressed version on the left and the compressed version on the right for each benchmark. On average, the uncompressed Parquet files have a slow down factor of 2.2 in Umbra. The compressed versions are, on average, 3.4× slower. Hyper is 10.2× slower than the corresponding database relation version for the compressed version. If we compare the execution times of the Hyper and Umbra Parquet scans with one another, we still achieve a speedup factor of 3.5× over Hyper. Compared to our black box evaluation, where we were 12× faster than Hyper, we can see that the statistics computation also added an average speedup of 3.5× to the overall query execution time.

#### 4.4 Database Relation Comparison

The main goal of adding the Parquet view functionality to our database system is to offer a reasonable alternative to database relations. The main advantage of database relations is

that they have additional time to get to know the data during the initialization phase. For a fair comparison, we, therefore, have to consider the time it takes to execute the *CREATE TABLE* and *INSERT* statements. We measure how often a query has to be executed to make it reasonable to preload the data into the database instead of working on the Parquet files:

$$db\_init + x \cdot db\_exec > pq\_first + (x - 1) \cdot pq\_exec \quad (1)$$

We use the TPC-H benchmark with scale factor 10 and the uncompressed Parquet files for this evaluation. To get the number of required repetitions, we have to solve Inequality 1. As long as the left side of the inequation takes longer than the right side, preloading the data is not beneficial. The left side consists of the execution time of the given query (*db\_exec*) multiplied by the number of repetitions (*x*) plus the time it takes to initialize all relations we require for the query (*db\_init*). The right side contains the execution time of the Parquet version (*pq\_exec*) times the number of repetitions (*x*) minus one. We add the first execution (*pq\_first*) separately since it contains the statistics computation overhead. All queries must be executed at least 30 times to amortize the initialization cost. On average, the queries would need to be executed more than 200 times to pay off the initialization and loading steps. Our evaluation shows that our framework presents a reasonable alternative to database relations with comparable performance and without initial loading costs.

## 5 Related Work

Most related work towards integrating raw data into query processing focuses on CSV files. Mühlbauer et al. [Mü13] bulk load CSV files in parallel by splitting the file into chunks, and utilize vectorization methods to speed up the loading process. Similar to our framework, they allow the user to query CSV files without initial loading. Nevertheless, missing metadata and the row-wise format limit the performance of CSV scans. The NoDB system *PostgresRAW* presented by Alagiannis et al. [Al12] identified and resolved these performance bottlenecks. Their adaptive indexing strategy collects metadata about the CSV files that improve future queries, such as the positions of attributes in the CSV file referenced in previous queries. They also work with the built-in statistics routines of Postgres, which are very limited, to improve the selectivity estimates of their query plans. Similar to our approach, they will only compute statistics for columns required by the current query. Olma et al. [Ol17] partition the underlying CSV file logically depending on the user's access patterns. Per partition, they store index structures, statistics, access frequencies and the average query selectivity to tune the partitions and indices continuously. While we add new estimates over time when new columns are accessed, the LEO optimizer presented by Stillger et al. [St01] introduces a feedback loop to incrementally fix estimates that are off by comparing them to the actual results.

Durner et al. [DLN21] deal with JSON files, where the main challenge is the lack of a schema and the possibility that the schema can change over time. Li et al. [Li20] build a storage engine based on the Apache Arrow file format. Their work focuses on improving the export

costs for data science and machine learning engines. Vogel et al. [Vo20] use the Parquet file format in their storage layer. Their main goal is to spread the data column-wise across a tierless device pool, depending on its usage. They split the Parquet files onto different devices to optimize the throughput. Idreos et al. [Id11], Abouzied et al. [AAS13], and Chenk et al. [CR14] incrementally improve query execution time on RAW files by loading more and more parts of the data into the database while executing queries. SAHARA[Br22], a table partitioning advisor, collects data access statistics and chooses a partitioning layout based on these statistics. Our synopses come in handy if the SAHARA optimizer is used since the data will be sorted by columns that are frequently used in the filter predicates.

The idea of synopses was used in the literature under different names. Moerkotte et al. [Mo98] called them small materialized aggregates (SMAs). They are used to store *min*, *max*, *count*, and *sum* aggregates on page-level granularity. Lang et al. [La16] extended these to PSMA (positional SMAs) with a lightweight index structure to narrow down scan ranges further. E3 (Eagle-Eyed Elephant) is the term Eltabakh et al. [El13] used to describe a set of techniques, namely range indices, inverted indices, materialized views, and a caching algorithm to prevent reading unnecessary data. Ziauddin et al. [Zi17] store minimum and maximum values of one or more columns over contiguous blocks called zone maps. Presto, introduced by Sethi et al. [Se19], is a distributed query engine that also uses the optional min/max statistics of Parquet files to skip pages and row groups if possible. The system has custom implementations, for example, for joins that work with the dictionary-encoded data to build hash tables, which is an interesting opportunity for future work. Armbrust et al. and Brehm et al. describe in their work [Ar20; Be22] how they process Parquet files. They can skip unneeded data more aggressively due to the clustering of Delta Lake. In addition, instead of collecting statistics to make future executions faster, Photon, their vectorized query engine for lakehouse environments, supports batch-level adaptivity and can switch execution kernels based on the metadata collected from previous batches.

## 6 Conclusion

Parquet files are great candidates for integration into the query processing of a database engine. In this paper, we introduced a framework that solves the challenges of this file format. We presented different techniques for making the performance of analytical workloads on Parquet files more convenient. We demonstrated that we achieve comparable execution times to database relations. Computing statistics while accessing the columns of a Parquet file for the first time introduces a small overhead, but the query plan optimizer benefits a lot from these. Furthermore, the synopses computation introduces another significant performance boost for Parquet files stored remotely. In addition, our approach can compete with existing Parquet scans and outperforms them in all presented scenarios.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725286). 

## References

- [AAS13] Abouzied, A.; Abadi, D. J.; Silberschatz, A.: Invisible loading: access-driven data transfer from raw files into database systems. In: EDBT. ACM, pp. 1–10, 2013.
- [AGN15] Abedjan, Z.; Golab, L.; Naumann, F.: Profiling relational data: a survey. *VLDB J.* 24/4, pp. 557–581, 2015.
- [Al12] Alagiannis, I.; Borovica, R.; Branco, M.; Idreos, S.; Ailamaki, A.: NoDB: efficient query execution on raw data files. In: SIGMOD Conference. ACM, pp. 241–252, 2012.
- [An94] Andersson, M.: Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In: ER. Vol. 881. Lecture Notes in Computer Science, Springer, pp. 403–419, 1994.
- [Ar20] Armbrust, M.; Das, T.; Paranjpye, S.; Xin, R.; Zhu, S.; Ghodsi, A.; Yavuz, B.; Murthy, M.; Torres, J.; Sun, L.; Boncz, P. A.; Mokhtar, M.; Hovell, H. V.; Ionescu, A.; Luszczak, A.; Switakowski, M.; Ueshin, T.; Li, X.; Szafranski, M.; Senster, P.; Zaharia, M.: Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. *Proc. VLDB Endow.* 13/12, pp. 3411–3424, 2020.
- [ASF13] Apache Software Foundation: Apache Parquet, 2013, URL: <https://parquet.apache.org>, visited on: 12/15/2022.
- [ASF14] Apache Software Foundation: Apache Spark, 2014, URL: <https://spark.apache.org/>, visited on: 12/15/2022.
- [ASF16] Apache Software Foundation: Apache Arrow, 2016, URL: <https://arrow.apache.org/>, visited on: 12/15/2022.
- [Be22] Behm, A.; Palkar, S.; Agarwal, U.; Armstrong, T.; Cashman, D.; Dave, A.; Greenstein, T.; Hovsepian, S.; Johnson, R.; Krishnan, A. S.; Leventis, P.; Luszczak, A.; Menon, P.; Mokhtar, M.; Pang, G.; Paranjpye, S.; Rahn, G.; Samwel, B.; van Bussel, T.; Hovell, H. V.; Xue, M.; Xin, R.; Zaharia, M.: Photon: A Fast Query Engine for Lakehouse Systems. In: SIGMOD Conference. ACM, pp. 2326–2339, 2022.
- [Br22] Brendle, M.; Weber, N.; Valiyev, M.; May, N.; Schulze, R.; Böhm, A.; Mörkotte, G.; Grossniklaus, M.: SAHARA: Memory Footprint Reduction of Cloud Databases with Automated Table Partitioning. In: EDBT. OpenProceedings.org, 1:13–1:26, 2022.
- [BZN05] Boncz, P. A.; Zukowski, M.; Nes, N.: MonetDB/X100: Hyper-Pipelining Query Execution. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), pp. 225–237, 2005.
- [CLP22] Crotty, A.; Leis, V.; Pavlo, A.: Are You Sure You Want to Use MMAP in Your Database Management System? In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2022.
- [CR14] Cheng, Y.; Rusu, F.: Parallel in-situ data processing with speculative loading. In: SIGMOD Conference. ACM, pp. 1287–1298, 2014.

- [De13] Dem, J. L.: Announcing Parquet 1.0: Columnar Storage for Hadoop, 2013, URL: [https://blog.twitter.com/engineering/en\\_us/a/2013/announcing-parquet-10-columnar-storage-for-hadoop](https://blog.twitter.com/engineering/en_us/a/2013/announcing-parquet-10-columnar-storage-for-hadoop), visited on: 12/15/2022.
- [DLN21] Durner, D.; Leis, V.; Neumann, T.: JSON Tiles: Fast Analytics on Semi-Structured Data. In: SIGMOD Conference. ACM, pp. 445–458, 2021.
- [El13] Eltabakh, M. Y.; Özcan, F.; Sismanis, Y.; Haas, P. J.; Pirahesh, H.; Vondrák, J.: Eagle-eyed elephant: split-oriented indexing in Hadoop. In: EDBT. ACM, pp. 89–100, 2013.
- [Fl07] Flajolet, P.; Fusy, É.; Gandouet, O.; Meunier, F.: Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: Discrete Mathematics and Theoretical Computer Science. Discrete Mathematics and Theoretical Computer Science, pp. 137–156, 2007.
- [FN19] Freitag, M.; Neumann, T.: Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2019.
- [GG11] Google: Snappy, a fast compressor/decompressor, 2011, URL: <https://github.com/google/snappy>, visited on: 12/15/2022.
- [Id11] Idreos, S.; Alagiannis, I.; Johnson, R.; Ailamaki, A.: Here are my Data Files. Here are my Queries. Where are my Results? In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), pp. 57–68, 2011.
- [JN20] Jiang, L.; Naumann, F.: Holistic primary key and foreign key detection. *J. Intell. Inf. Syst.* 54/3, pp. 439–461, 2020.
- [Ka14] Karpathiotakis, M.; Branco, M.; Alagiannis, I.; Ailamaki, A.: Adaptive Query Processing on RAW Data. *Proc. VLDB Endow.* 7/12, pp. 1119–1130, 2014.
- [KN11] Kemper, A.; Neumann, T.: HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: ICDE. IEEE Computer Society, pp. 195–206, 2011.
- [La16] Lang, H.; Mühlbauer, T.; Funke, F.; Boncz, P. A.; and Alfons Kemper, T. N.: Data Blocks: Hybrid OLTP and OLAP on Compressed Storage using both Vectorization and Compilation. In: SIGMOD Conference. ACM, pp. 311–326, 2016.
- [Le14] Leis, V.; Boncz, P. A.; Kemper, A.; Neumann, T.: Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In: SIGMOD Conference. ACM, pp. 743–754, 2014.
- [Le15] Leis, V.; Gubichev, A.; Mirchev, A.; Boncz, P. A.; Kemper, A.; Neumann, T.: How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9/3, pp. 204–215, 2015.
- [Le18] Leis, V.; Haubenschild, M.; Kemper, A.; Neumann, T.: LeanStore: In-Memory Data Management beyond Main Memory. In: ICDE. IEEE Computer Society, pp. 185–196, 2018.

- [Li20] Li, T.; Butrovich, M.; Ngom, A.; Lim, W. S.; McKinney, W.; Pavlo, A.: Mainlining Databases: Supporting Fast Transactional Workloads on Universal Columnar Data File Formats. *Proc. VLDB Endow.* 14/4, pp. 534–546, 2020.
- [Me10] Melnik, S.; Gubarev, A.; Long, J. J.; Romer, G.; Shivakumar, S.; Tolton, M.; Vassilakis, T.: Dremel: Interactive Analysis of Web-Scale Datasets. *Proc. VLDB Endow.* 3/1, pp. 330–339, 2010.
- [MK17] Motl, J.; Kordík, P.: Foreign Key Constraint Identification in Relational Databases. In: *ITAT*. Vol. 1885. CEUR Workshop Proceedings, CEUR-WS.org, pp. 106–111, 2017.
- [MNS09] Moerkotte, G.; Neumann, T.; Steidl, G.: Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *Proc. VLDB Endow.* 2/1, pp. 982–993, 2009.
- [Mo98] Moerkotte, G.: Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. In: *VLDB*. Morgan Kaufmann, pp. 476–487, 1998.
- [Mü13] Mühlbauer, T.; Rödiger, W.; Seilbeck, R.; Reiser, A.; Kemper, A.; Neumann, T.: Instant Loading for Main Memory Databases. *Proc. VLDB Endow.* 6/14, pp. 1702–1713, 2013.
- [Ne11] Neumann, T.: Efficiently Compiling Efficient Query Plans for Modern Hardware. *Proc. VLDB Endow.* 4/9, pp. 539–550, 2011.
- [NF20] Neumann, T.; Freitag, M.: Umbra: A Disk-Based System with In-Memory Performance. In: *CIDR*. [www.cidrdb.org](http://www.cidrdb.org), 2020.
- [OI17] Olma, M.; Karpathiotakis, M.; Alagiannis, I.; Athanassoulis, M.; Ailamaki, A.: Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing. *Proc. VLDB Endow.* 10/10, pp. 1106–1117, 2017.
- [PN17] Papenbrock, T.; Naumann, F.: Data-driven Schema Normalization. In: *EDBT*. [OpenProceedings.org](http://OpenProceedings.org), pp. 342–353, 2017.
- [RM19] Raasveldt, M.; Mühleisen, H.: DuckDB: an Embeddable Analytical Database. In: *SIGMOD Conference*. ACM, pp. 1981–1984, 2019.
- [SE19] Skyscanner Engineering: Building a self-served ETL pipeline for third-party data ingestion, 2019, URL: <https://medium.com/@SkyscannerEng/building-a-self-served-etl-pipeline-for-third-party-data-ingestion-3959eab6840b>, visited on: 12/15/2022.
- [Se19] Sethi, R.; Traverso, M.; Sundstrom, D.; Phillips, D.; Xie, W.; Sun, Y.; Yegitbasi, N.; Jin, H.; Hwang, E.; Shingte, N.; Berner, C.: Presto: SQL on Everything. In: *ICDE*. IEEE, pp. 1802–1813, 2019.
- [Sh99] Shanmugasundaram, J.; Tufte, K.; Zhang, C.; He, G.; DeWitt, D. J.; Naughton, J. F.: Relational Databases for Querying XML Documents: Limitations and Opportunities. In: *VLDB*. Morgan Kaufmann, pp. 302–314, 1999.

- [St01] Stillger, M.; Lohman, G. M.; Markl, V.; Kandil, M.: LEO - DB2's LEarning Optimizer. In: VLDB. Morgan Kaufmann, pp. 19–28, 2001.
- [TSF20] Trino Software Foundation: Trino, 2020, URL: <https://trino.io/>, visited on: 12/15/2022.
- [Vo18] Vogelsgesang, A.; Haubenschild, M.; Finis, J.; Kemper, A.; Leis, V.; Mühlbauer, T.; Neumann, T.; Then, M.: Get Real: How Benchmarks Fail to Represent the Real World. In: DBTest@SIGMOD. ACM, 1:1–1:6, 2018.
- [Vo20] Vogel, L.; van Renen, A.; Imamura, S.; Leis, V.; Neumann, T.; Kemper, A.: Mosaic: A Budget-Conscious Storage Engine for Relational Database Systems. Proc. VLDB Endow. 13/11, pp. 2662–2675, 2020.
- [WG17] Weeks, D.; Gianos, T.: Petabytes Scale Analytics Infrastructure @Netflix, 2017, URL: <https://www.infoq.com/presentations/netflix-big-data-infrastructure/>, visited on: 12/15/2022.
- [Zi17] Ziauddin, M.; Witkowski, A.; Kim, Y. J.; Lahorani, J.; Potapov, D.; Krishna, M.: Dimensions Based Data Clustering and Zone Maps. Proc. VLDB Endow. 10/12, pp. 1622–1633, 2017.

# The Evolution of LeanStore

Adnan Alhomssi<sup>1</sup>, Michael Haubenschild<sup>2</sup>, Viktor Leis<sup>3</sup>

**Abstract:** LeanStore is a high-performance key-value storage engine optimized for many-core processors and NVMe SSDs. This paper provides the first full system overview of all LeanStore components, several of which have not yet been described. We also discuss crucial implementation details, and the evolution of the entire system towards a design that is both simple and efficient.

**Keywords:** storage engine; LeanStore; B-tree; caching; SSD; multi-core

## 1 Introduction

**In-Memory DBMS.** For more than a decade, main-memory database systems have been the focal point of research on high-performance transaction processing. Academically, this research program has been a tremendous success, introducing innovative techniques and achieving unprecedented performance results. However, the real-world adoption of pure in-memory database system has been limited. It is fair to say that even the most successful in-memory systems such as VoltDB [SW13], Hekaton [Di13], and HANA [Fä11] remain niche products. General-purpose transaction processing is still dominated by traditional (disk-based) database systems such as Oracle, SQL Server, and their cloud-native cousins Aurora [Ve17] and Socrates [An19].

**Storage Cost Trends.** The main-memory revolution was primarily fueled by rapidly shrinking DRAM prices. From 2000 to 2012 the price/byte for DRAM dropped by about 300× [HHL20]. This made it feasible to keep non-trivial databases in main memory. After 2012, however, DRAM prices have largely *stagnated* [HHL20]. And with Intel canceling Optane in 2022, persistent memory failed to achieve its promise as a replacement for traditional storage technologies.

**Flash to the Rescue.** In contrast to DRAM, the last decade has seen dramatic price reductions for flash. In 2005, the cost per byte for DRAM and for flash was comparable. Today, flash is about 20–50× cheaper [HHL20]. And flash SSDs have not just become cheap, with the introduction of NVMe they have also become very fast. For example, a Samsung PM1733 PCIe 4.0 SSD achieves up to 7 GB/s read throughput and 1.5 million random reads per second [Sa22a]. Commodity servers have dozens of PCIe lanes, which means that hardware

---

<sup>1</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg, adnan.alhomssi@fau.de

<sup>2</sup> Salesforce, mhaubenschild@salesforce.com

<sup>3</sup> Technische Universität München, leis@in.tum.de

setups with 8 or 10 NVMe SSDs have become common (see Azure’s Lsv2 [AW22b] and EC2’s i3en [AW22a] instances). Consequently, servers with 10+ million I/O operations per second and an aggregated bandwidth rivaling DRAM are not just possible – but affordable and readily available. And the trend will continue with the upcoming PCIe 5.0 servers and SSDs [Sa22b], which will almost double I/O bandwidth once more.

**NVMe-Optimized Systems.** Neither in-memory nor disk-based systems even come close to being able of exploiting the capabilities of modern SSDs. Many of the design decisions of in-memory systems (e.g., small index nodes) are simply not suitable for storage on flash. Existing disk-based systems, on the other hand, look more conceptually promising (e.g., page-based data organization), but were developed when storage was slower by several orders of magnitude. As a result, a system such as PostgreSQL is completely *CPU*-bound on out-of-memory workloads on NVMe SSDs [HHL20]. Even though they are often treated that way, SSDs are not just fast disks – for good performance they require new DBMS designs.

**LeanStore.** The LeanStore [Le18] project started in 2016. The goal was to show that one can achieve performance close to in-memory systems without having to keep all data in memory. To be efficient and robust on modern hardware, LeanStore combines many of the modern in-memory optimizations (e.g., CPU and cache efficiency, lightweight synchronization [LHN19], contention avoidance [AL21]) with traditional decades-old DBMS techniques (e.g., buffer management [Le18], B-trees, paged storage, physiological logging, fuzzy checkpoints [Ha20]).

**This Paper.** While many of these novel techniques have already been published in separate papers [AL21; Ha20; Le18; LHN19] (or will be published in upcoming papers [AL23]), this paper provides the first overview of the full LeanStore system. Because research papers usually focus on a single contribution rather than the full system and the interaction of components, this includes many crucial unpublished aspects – such as the B-tree design, synchronization primitives, and logging optimizations. Doing so, we also describe the evolution of the system, which changed considerably over its lifetime. Many original design decisions had to be reconsidered in the light of hardware evolution, and over time we managed to radically simplify important implementation details. For example, the original paper [Le18] argues that a single latch for managing I/O operations is fast enough, and describes epoch-based memory reclamation. The former quickly proved wrong with fast SSDs, while the latter proved to be completely unnecessary.

**Outline.** The rest of the paper is organized as follows. Sect. 2 discusses related work. Sect. 3 then provides an overview of the functionality and key components of LeanStore. We then focus on four components, namely the B-tree in Sect. 4, the synchronization primitives in Sect. 5, the buffer manager in Sect. 6, and logging in Sect. 7. Sect. 8 experimentally compares LeanStore with two state-of-the-art storage engines for both in-memory and out-of-memory workloads. Finally, we summarize the paper in Sect. 9.

## 2 Related Work

Surprisingly, in the past decade research on designing flash-optimized database systems has been fairly sparse. Most of the new system developments focused on designs optimized for main memory (e.g., VoltDB [SW13], Hekaton [Di13], HANA [Fä11], HyPer [Ke12], Silo [Tu13]) or persistent memory. A notable exception is the Umbra [NF20] system, which adopted LeanStore’s swizzling-based buffer management design, but extends it with support for variable-sized pages. There are two state-of-the-art open source storage engines optimized for flash that should be mentioned. WiredTiger [Mo22], which is the default storage engine of MongoDB, has several conceptual similarities with LeanStore, including reliance on B-trees and pointer swizzling. RocksDB [Do21], in contrast, is the most prominent LSM-based storage engine, optimizing for lower write amplification rather than read performance. We experimentally compare with both systems in Sect. 8 and descriptively in the remaining of this section.

Unlike LeanStore, **WiredTiger** uses a separate representation for on-disk and in-memory B-Tree nodes. In-memory nodes are dynamically allocated and not hosted on a fixed-size page like traditional buffer-managed systems do. On-disk nodes images are stored in immutable block that are grouped in files. In-memory node use prefix compression while on-disk blocks are mostly compressed with snappy algorithm. An index operation follows virtual memory pointers until it hits a non-resident (unswizzled) node. In this case, it reads the file and builds an in-memory node out of it. Updating a key in a leaf inserts a new entry in the node’s per-key skip list. When the node is full and it is time to evict, WiredTiger reconciles, i.e., serializes, the node and merges all updates in the node to create the disk image. In LeanStore, we update nodes in-place and write the in-memory node as it is on SSDs without any serialization overhead.

To synchronize indexes, WiredTiger uses lock-free algorithms where LeanStore uses optimistic lock coupling. To make sure a thread is following a valid pointer, WiredTiger pushes the node’s pointer to a hazard list before it accesses it. Nodes pointed by this list are excluded from memory reclamation or eviction. The processing thread removes the pointer once it leaves the node. Compared to LeanStore’s optimistic lock coupling and pointer swizzling, hazard pointers in WiredTiger cost one more memory flush (fence) to publish the new hazard pointer for each node access.

**RocksDB** relies on the Log-Structured Merge-tree (LSM) data structure to reduce (random) write and space amplification and prolong the lifetime of SSDs which have limited endurance. By choosing B-trees as the foundational data structure, LeanStore is a more read-optimized design. Nevertheless, our design reduces write amplification by using 4 KiB pages and increases space utilization for B-Trees through the XMerge [AL21] technique that optimistically merges under-fill neighboring nodes.

The first layer “MemTable” in the LSM resides in memory and absorbs all write operations. Once it reaches a configured size, RocksDB flushes its content to an immutable Sorted

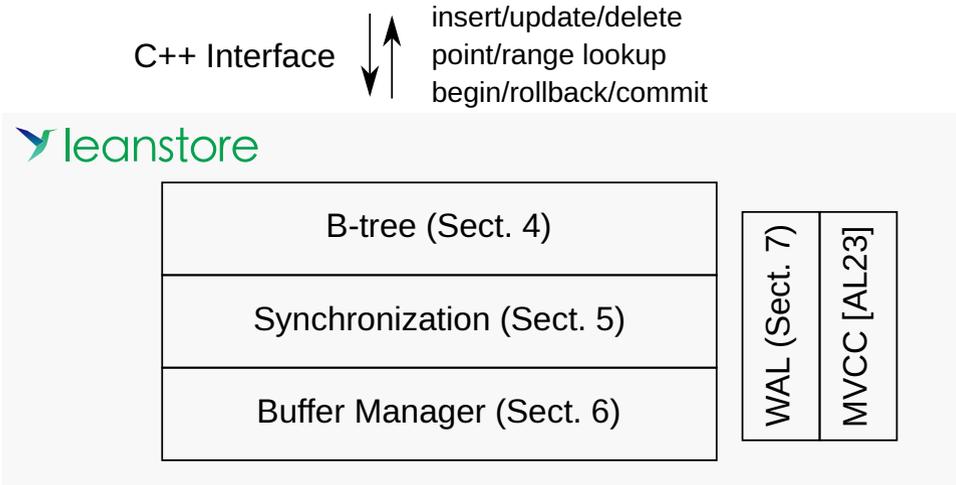


Fig. 1: Overview of main system components

String Table (SSTable). The earlier versions of RocksDB only supported single-writer to the MemTable. The newer ones allow multiple writers but only for the skip-based implementation. This signals how multi-core CPU support came a second thought where LeanStore is built up from the ground to scale on many-core with optimistic lock coupling.

To cache SST blocks, RocksDB relies on third-party allocator to manage its buffer pool instead from implementing its own. Concretely, RocksDB uses jemalloc to cache KV pairs from SSTs in variable-length blocks [Do21]. However, this reliance on jemalloc leads to fragmentation and performance issues that differ between workloads and instances. RocksDB user has to set two memory budgets. The first one for the size of the first LSM in-memory layer (called MemTable) – which also impacts the compaction strategy. The second one for caching which should account for the overhead by the allocator. The complex design and the resulting different configuration parameters made Tuning memory configuration is one of the LeanStore design does not suffer from this complication. We use a single buffer pool with exact capacity and fixed-size pages to host all kind of nodes.

### 3 LeanStore System Overview

**Functionality and System Overview.** LeanStore is a storage engine supporting basic key value operations (insert/update/delete, point/range lookup) and transactional semantics (begin, commit, rollback). This functionality is exposed through a C++ interface, i.e., as an embeddable library rather than a server process. Systems with comparable functionality include RocksDB and WiredTiger, which are often used as building blocks for full-blown

data management systems. Fig. 1 illustrates the main components and the layering of the system. In the following, we provide a high-level overview of each component.

**B-tree.** The main data structure in LeanStore for both indexing and storing data is a B<sup>+</sup>-tree. Keys and values are opaque byte sequences and the application is responsible for transforming and interpreting the data<sup>4</sup>. Although B-trees are not as fast [Wa18] as pure in-memory data structures such as ART [LKN13] or HOT [Bi22], they offer reasonable in-memory performance and work well in most out-of-memory situations. We implement several optimizations to narrow this gap to in-memory data structures as described in Sect. 4. In comparison with Log-Structured Merge (LSM) trees, which offer lower write amplification, B-trees are faster for reads and are simpler to implement while having far fewer parameters. Nevertheless, it is possible to offer LSM-trees in LeanStore as an additional option, and we already experimented with a prototype LSM implementation.

**Synchronization Primitives.** Modern CPUs have dozens of cores, which means that overall performance is often determined by scalability rather than single-threaded performance. In LeanStore, we implement an innovative synchronization framework that is scalable, has low overhead, and that is generic and easy to use. The synchronization primitives and their implementation is described in Sect. 5.

**Buffer Manager.** The first LeanStore paper [Le18] demonstrated that it is possible to implement buffer management with very little overhead in comparison with in-memory systems. The key ideas of that paper are pointer swizzling [Gr14] and a lightweight replacement strategy. With pointer swizzling, cached pages are directly accessible through pointers – avoiding any indirection data structure necessary in traditional buffer manager designs. An access to swizzled pages also does not incur any overhead for the page replacement algorithm. Once the system is running low on free memory, candidate pages for eviction are randomly unswizzled and put into a *cooling stage*. Two things can happen now: a candidate page is either fairly hot, in which case it will be swizzled again (without incurring I/O); or it is cold, in which case it is eventually evicted once it reaches the end of a FIFO list. Sect. 6 describes how we have significantly simplified page eviction and how we optimized I/O management to catch up with the increasing SSD performance.

**Snapshot Isolation and MVCC.** LeanStore supports snapshot isolation through an implementation of Multi-Version Concurrency Control (MVCC). Surprisingly, we found that the OLTP performance of state-of-the-art out-of-memory MVCC schemes collapses in the presence of long-running OLAP queries (and vice versa). In an upcoming paper [AL23], we propose a design that solves this problem. To do this, we combine (1) a novel out-of-memory commit protocol that enables efficient fine-granular garbage collection, (2) an auxiliary data

<sup>4</sup> LeanStore stores both keys and values/payloads as is. For payloads this is not a problem, but it presents an additional challenge for keys. For example, if we have an integer key and store it in the index as signed big endian, range scans will not yield the expected order. To solve this, it is necessary to transform keys (e.g., swap bytes for little endian). This is a widely-used technique that Graefe [Gr11] calls *normalized keys*.

Tab. 1: The motivation behind the design decisions and techniques behind LeanStore.

| Technique                   | Motivation   | Technique             | Motivation          |
|-----------------------------|--------------|-----------------------|---------------------|
| Pointer Swizzling [Le18]    | Large Buffer | Partitioned I/O Stage | NVMe IOPS           |
| Optimistic Locking [LHN19]  | Scale Reads  | Batched Async Writes  | NVMe IOPS           |
| Contention Split [AL21]     | Scale Writes | Group Commit          | NVMe IOPS           |
| Distributed Logging [Ha20]  | Scale TXs    | XMerge [AL21]         | Space Utilization   |
| Tuple-Wise Depend. Tracking | Scale TXs    | 4 KiB Pages           | Write Amplification |

structure that moves logically-deleted tuples out of the way of operational transactions, and (3) an adaptive version storage scheme.

**Logging.** In disk-based database systems, transactional durability is guaranteed by a write-ahead log (WAL). Because the WAL is usually a centralized data structure, it becomes a point of contention that prevents multi-core scalability. In a SIGMOD paper from 2020 [Ha20], we propose a solution that uses one WAL per thread, and that ensures correctness through a lightweight page-based tracking mechanism called Remote Flush Avoidance (RFA). Since then, we refined this scheme to further reduce unnecessary log flushes as described in Sect. 7.

In Tab. 1, we list the design decisions and techniques we have implemented in LeanStore as a response to the different hardware trends like large DRAM buffers, NVMe SSDs and many-core CPUs. The ones without citations are presented in this paper.

## 4 B<sup>+</sup>-Tree

Based on a classic slotted page B-tree node layout, we implement a number of B-tree optimizations that have been proposed in the literature.

**Node Layout.** Fig. 2 shows the layout of a leaf node storing three keys (“http://fau.de”, “http://tum.de”, and “http://uni-jena.de”). The slots at the front of the page contain the offset and length of the key/payload stored at the end of the page. An inner node looks the same except that it stores child pointers as a payload instead of values.

**Fence Keys and Prefix Compression.** Fence keys [Gr04] simplify the implementation of prefix compression, and they are stored in every node. They are immutable and determined on node initialization, i.e., after a split or merge. The lower and upper fence values are the exclusive lower and the inclusive upper bound for the keys that could be potentially stored in that node. When a new B-tree is created, the initial leaf page conceptually has its fence keys set to special values  $-\infty$  and  $\infty$ . All other fence keys are given automatically by the split (or

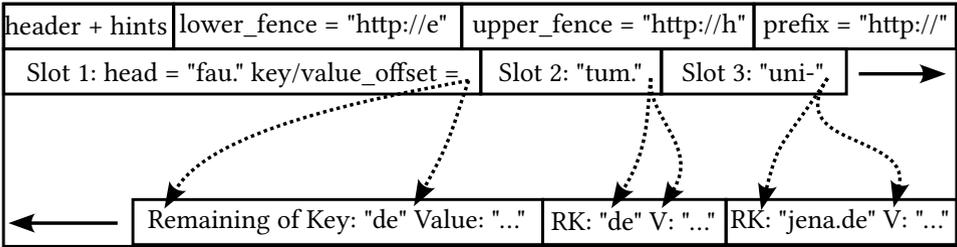


Fig. 2: Our variable-length keys and values B<sup>+</sup>-tree features fences, prefix compression, heads extraction and hints to accelerate binary search and save space

“separator”) keys during node splits. Using the fence keys, it becomes straightforward to implement the well known key prefix compression technique [BU77]. The prefix is derived from the fence keys by taking the longest common prefix of both fence keys.

**Speeding Up Binary Search with Heads.** Although the basic slotted page layout enables binary search, it does so quite inefficiently because every key comparison will require dereferencing a pointer to obtain the key – leading to many cache misses. To reduce the number of cache misses, Graefe and Larson proposed “poor man normalized keys” [GL01], which are a fixed-size prefix of the key. In each key slot, we therefore store the first 4 bytes of the key in the equivalent unsigned integer representation and call it *head*. By default, all keys are stored as strings and we use lexicographical order to compare keys. That means on little-endian systems (the majority of today’s CPU architectures), we fold integer and compound keys using byte swap operations to maintain the same sort order for their serialized string representation. This allows efficient less-than comparisons based on the integer representation. With the heads available, we can first perform binary search solely on the heads in the slots array using cheap integer comparison. Only for heads that equal the lookup key, it becomes necessary to retrieve the full key and perform a byte-wise comparison.

**Avoiding Binary Search with Hints.** To accelerate binary search in B-Tree nodes, we further implement the *hints* optimization, which can be considered a form of in-page micro-index [Lo01]. Instead of starting the binary search over the whole range, i.e., [0, #slots], we try to first shrink the range using cache efficient search over hints. These hints are stored in a dense array with a fixed number of entries (we use  $\#hints = 16$ ). The distance between keys from which the heads are taken is constant and defined as  $distance = \#slots \div (\#hints + 1)$ . The original position of each hint key is determined using a simple formula. As shown in List. 1, after finding the candidate range in hints, we can translate it to the corresponding range in the node keys’ slots array. We also store just a copy of the first four bytes (*hint\_size*) instead of the whole key in the hints array. Because all hints fit in a single cache line, we can perform a quick binary search using integer comparisons to minimize the candidate range (lower, upper) for the key we are looking for. We also found updating the hints structure to have negligible overhead.

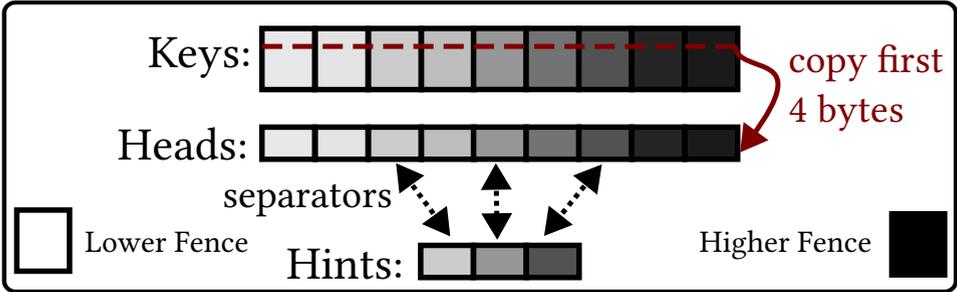


Fig. 3: Hints optimization: the heads of equally-distanced keys are stored as 4-byte integers, inner node binary search starts by narrowing the range using quick binary search over hints

```

updateHints() { // hints_count = 16, hint_size = 4 -> 64 Bytes
    u32 dist = slots_count / (hints_count + 1);
    for (u16 i = 0; i < hints_count; i++)
        hint[i] = key[dist * (i + 1)].substr(0, hint_size);
}

searchHints(u8* key, u16 key_length) { // Binary search starts with returned range
    u32 key_head = head(key, key_length); // first hint_size bytes of key
    u32 pos, pos2;
    for (pos = 0; pos < hint_count; pos++) // skip smaller hints
        if (hint[pos] >= key_head)
            break;
    for (pos2 = pos; pos2 < hint_count; pos2++) // find equal hints
        if (hint[pos2] != key_head)
            break;
    // convert pos and pos2 to full key range
    u32 dist = slots_count / (hints_count + 1);
    u32 lower = pos * dist;
    u32 upper = (pos2 < hint_count) ? ((pos2 + 1) * dist) : slots_count;
    return {lower, upper};
}

```

List. 1: Narrowing binary search range using efficient integer search over hints

**Contention Split and XMerge.** In contrast to deterministic data structures such as tries, the same set of keys may result in different B-tree structures (usually only depending on the insertion order). We exploit this observation by dynamically and adaptively optimizing the B-tree structure using the *Contention Split* and *XMerge* optimizations. *Contention Split* reduces unnecessary latch contention (i.e., where a page contains more than one hot tuple), by splitting the node – even though the node may not be full. The second optimization, increases the fill factor of B-tree nodes by opportunistically merging  $X$  neighboring nodes into  $X-1$  nodes. Both optimizations have been described in detail previously [AL21].

## 5 Practical and Scalable Synchronization

### 5.1 Optimistic Lock Coupling and Buffer Management in C++

With optimistic latching, we eliminated unnecessary read contention for short page reads [Le16]. The original implementation relied on a single atomic version counter and a spin lock implementation. This simple design is not robust enough to handle the variety of workloads that a database has to deal with [Bö20]. Reverse CPU priority and unfairness are examples for anomalies that such spins locks exhibit.

**Hybrid Latches.** In the latest iteration of the LeanStore design, we use hybrid latches with OS support that allow a “pessimistic shared” and an “exclusive” mode besides the original optimistic mode. We implemented them using a `std::shared_mutex` next to a `std::atomic<uint64_t>` which serves as the version counter. With the three options in place, we can use the most suitable one for each operation: For long page reads, e.g., in scans, we directly use the pessimistic shared mode, which also simplifies the scan logic compared to our original implementation. Before, we had to worry about concurrent modifications could lead to restarts while scanning a page. For short reads in leaf nodes and for inner node traversal, we first try to latch optimistically and immediately fallback to the pessimistic shared mode if we find the latch is already locked. Page modification operations, i.e., insert, update, and delete, latch the page in exclusive mode.

**Page Guard.** To help with the complexity of buffer management and the different latching modes, we use Page Guards to abstract some of the complexity away from the data structure implementation. Page Guards are a C++ template class that handle latching and buffer management and provide a simple interface to implement a data structure with. They follow the known Resource acquisition is initialization (RAII) programming idiom. Once a page guard is created on the stack in certain latch mode, it keeps the page latched in that mode until the guard object goes out of scope. This protects against leaking latches for pages that we forget to unlock in Code. We also use the page guard to access the underlying data structure by overloading the “->” operator, which is helpful to make sure we are accessing a latched page. In List. 2, we show how we use page guards to implement the B-tree insert method. For tree traversal, we use the optimistic guard to avoid contention on inner nodes. Once we locate the required leaf node, we upgrade to exclusive mode by constructing an `ExclusivePageGuard` from the moved `OptimisticPageGuard`. The overloaded arrow operator (->) returns the underlying buffer managed object which is the `BTreeNode` in our case.

### 5.2 Implementing Restart in C++

One consequence of our optimistic latch implementation and the fact that we do not hold a latch on a parent page while reading one of its children from storage is that we have to make data structure operations restartable. Unlike optimistic concurrency control, where

```
void insert(key, value) {
    while(true) { // restart until success
        jumpmuTry() {
            // Traverse inner nodes optimistically
            OptimisticPageGuard<BTreeNode> p_opt_guard;
            // Swizzle the root_swip (load the page) if necessary and
            // optimistically latch the page (load the version).
            // It can trigger restart if root_latch version has changed in-between
            OptimisticPageGuard<BTreeNode> c_opt_guard(root_swip, root_latch);
            while(c_opt_guard->is_inner) {
                // Binary search to find the child swip
                Swip &c_swip = c_opt_guard->searchInner(key);
                p_opt_guard = std::move(c_opt_guard); // reassigns without releasing latch
                // Checks if p_opt_guard has been modified before following c_swip
                c_opt_guard = OptimisticPageGuard(p_opt_guard, c_swip);
            }
            // Latch the leaf in pessimistic exclusive mode
            ExclusivePageGuard c_ex_guard(std::move(c_opt_guard));
            p_opt_guard.release(); // parent is not needed anymore
            if(c_ex_guard->canInsert(key,value)) {
                c_ex_guard->insert(key, value);
                jumpmu_return; // success
            } else { // Leaf is full. We can't immediately split because
                // we released the parent. Thus, we copy the inclusive
                // upper bound and release all latches.
                upper_bound = c_ex_guard->upper_bound;
                c_ex_guard.release();
                // Then find and split the leaf that covers given upper bound
                // starting from the root
                trySplit(upper_bound); // Next iteration will find space in leaf
            }
        } jumpmuCatch() {} // Jumps are handled simply by restarting from root
    }
}
```

List. 2: B-tree Insert Code Illustrating Page Guard Usage

we potentially restarts the whole transaction at the end, low-level optimistic latching can force the index operation to restart any point in the code. This makes optimistic latching more challenging to adopt. In the following, we discuss the options in C and C++ that one could use to implement such restarts. Then we discuss our custom solution *JumpMU* that allows us to implement scalable and efficient restarts while maintaining readable code.

**Goto and Labels.** The *goto* statement immediately transfers control to another statement in the same function. When we detect a concurrent write to the page we optimistically read from, we can *goto* back to the first statement in our data structure operation. However, this works only in the same stack frame (same function call in C++). Storage engines are usually complex and are composed out of several components like a buffer manager and a B-tree with several functions calling each other. With *goto*, a function call has to inform its caller if it returned successfully or faced a synchronization error that forces a restart. Integrating these checks everywhere in the code wherever an optimistic read could occur is very wearisome, error-prone, and makes the code hard to read.

**C++ Exceptions.** Another native option in C++ is using *exceptions*. Exceptions at first sight look like a very good fit. They support jumping up several stack levels to the place where the data structure started its operation. Also, they automatically unwind the stack and call the destructors along the path to the exception handler. However, the standard exceptions implementation does not scale with many CPU cores, because common implementations (GCC and LLVM) acquire a global lock every time an exception is thrown. Although the scalability issue can be mitigated by avoiding the global lock that protects the list of dynamically loaded objects, the CPU cost of handling an exception remains high. A C++ exception is handled in two phases. The first phase traverses the stack up until it finds a landing pad that handles the thrown exception. The second phase starts from scratch and unwinds each call stack on the way up to the exception handler. This process costs around 10K instructions and lies on the hot path of B-tree accesses in our engine. For instance, if the exclusive lock acquisition for a child node failed after we managed to lock the parent, then the restart process will be triggered after around 10K instructions. During this time, the parent remains exclusively locked which also affects scalability besides being inefficient.

**Long Jump in C.** The C standard library provides a mechanism to change control flow back to a certain checkpoint set by the user across multiple levels of call stacks. The function *setjmp* sets this checkpoint which can be used later as a landing pad by the complementary function *longjmp*. For storage engines written in C, this mechanism is all what one could need to implement restarts efficiently. However, the correct stack unwinding that C++ takes care of automatically for the programmer does not happen with *longjmp*. This means that none of the destructors of the objects allocated on the stack between *longjmp* and *setjmp* will be called, which can lead to resource leaks and many other problems. Thus, this mechanism alone is no viable option either.

**JumpMU: Long Jump with Manual Stack Unwinding.** To implement restarts efficiently, we propose a solution that combines the benefits of C++ exceptions and the C *longjmp*

function at the cost of additional destructor tracking that could be also automated. We call our solution *JumpMU* where MU stands for manual unwinding. JumpMU builds on `setjmp` and `longjmp` and provides a C++ safe variant by manually calling all non-trivial destructors that the C `longjmp` alone would. Concretely, just before `longjmp` is called, JumpMU destructs all statically created objects on the stack since the last `setjmp` – effectively unwinding the stack. Objects without non-trivial destructors are irrelevant for JumpMU. To achieve this efficiently, we split the work between compile-time and run-time. At run-time, we maintain a stack of two pointers: one to the object we destruct when we jump and one to its destructor function. The order in which the pointers pair is inserted corresponds to the order of objects construction on the thread stack. An object constructor pushes the needed pointers to destruct it into JumpMU stack and its destructor pops it from the stack. At compile-time, we augment the constructors and destructors with the necessary instructions to keep the JumpMU stack in sync, i.e., push and pop pointers.

**JumpMU: Interface.** We provide a similar programming experience to C++ exceptions by using C macros as shown in the example in List. 2. Similar to the C++ `try` keyword, we use the `jumpmtry()` to define a code block that can trigger a restart. Implementation wise, `jumpmtry` calls `setjmp` and saves the current environment in a thread local array that holds multiple environment structs so we can nest `setjmps`. Once a jump is triggered – by calling `jumpmu::jump()`, the control is transferred to the block defined in `jumpmuCatch()`.

**JumpMU: Current Caveats.** When we force the control flow out of a `jumpmtry` block, we have to care of destroying the `setjmp` environment that was created by the `jumpmtry` block. For now, we use extra macros for every keyword that could move the control flow out of the surrounding `try` block. This includes `jumpmu_continue`, `jumpmu_break` and `jumpmu_return`. Moreover, the current implementation requires the user to manually annotate all objects definition that lie in a jump range with JumpMU macros. All of the above can be automated using a compiler plugin, which we leave for a future work.

## 6 Buffer Manager

In this section, we describe the changes that our buffer manager design witnessed since its introduction in the original LeanStore paper [Le18]. The changes are geared towards simplification and better scalability. Although our initial designed delivers very competitive in-memory performance that is similar to an in-memory system, its implementation remained complicated. We revised our design and managed to simplify it significantly as we show in the following.

### 6.1 Memory Reclamation is Unnecessary

In the original LeanStore design, we implemented an epoch-based memory reclamation. The epoch-based approach was not only invasive and complex to maintain, it also lacked

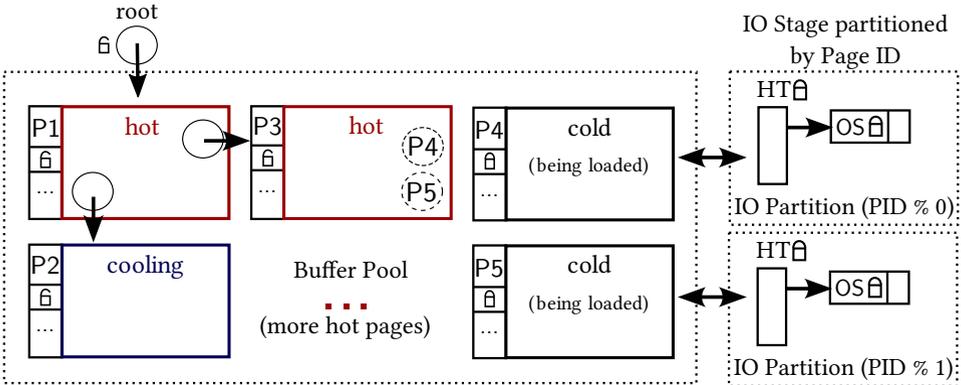


Fig. 4: The revised buffer manager design obliterates the need for a cooling stage and partitions the I/O stage by the page id to avoid contention on the hash table latch

robustness because a single slow thread could impede the advancement of the current global epoch which hinders the eviction of pages. Surprisingly, although memory reclamation seems mandatory for optimistic and lock-free synchronization, we eventually realized that this component is not needed after all. Thus, we could simplify our system by ensuring that 1) we do not reset or change the address of the atomic version counter of any buffer frame and 2) we do not release the memory allocated to the buffer back to the operating system. Condition 1 can easily be satisfied, and condition 2 is generally true in buffer managers. Thus, memory reclamation is completely unnecessary. We consequently we removed the code and got rid of the robustness issues of epoch-based reclamation.

## 6.2 Page Replacement

**Lightweight Replacement Strategy Without a Cooling Stage.** To achieve near in-memory performance, we aim at removing any overhead from the hot path, i.e., from read-only access to frequently visited pages. To that end, we designed our replacement strategy right from the start to work differently than in a traditional buffer manager. Instead of tracking the hot set of accessed pages, we identify the cold set of unused pages in a background fashion. Our first proposal *LeanEvict* follows this design principle [Le18], but did so using an unnecessarily complicated *cooling stage*, which was also vulnerable to contention. In this paper, we propose replacing *LeanEvict* with the well-known Second Chance replacement strategy. Second Chance does not add any overhead to the hot path and works in the same background fashion as *LeanEvict*. At the same time experimental results show that it does not lead to worse eviction candidates than the much more elaborate *LeanEvict*. In the following we describe our implementation of Second Chance in LeanStore.

**Swizzling-Based Second Chance Implementation.** Whenever free pages are needed, we pick a set of 64 random buffer frames and load their status. Then, we iterate over them and

“cool the hot” (swizzled) buffer frames by 1) setting the most significant bit in the parent’s swip that points to them 2) changing their buffer frame status to “cool”. Any page that is already cool gets written back to disk if dirty, or evicted immediately otherwise. Note that with second-chance, the whole cooling stage, which includes a hash table and FIFO queue, is not needed anymore. Moreover, a cool swip holds a virtual memory pointer also when it is cool but the pointer is tagged to mark the page it points to as cool. This is in contrast to our previous design where a swip to a cool page gets unswizzled and replaced by the page identifier. Accessing a cool page does not require any access to the cooling hash table as is the case in *LeanEvict*. A hot page access does not require any extra work with our Second Chance implementation because we encode the status of the child node in the swip that points to it. If the cooling bit is not set, then it follows the swip pointer as usual. Otherwise, in the cold page access case, the bit is unset in the swip and the buffer frame status is changed back to hot.

**No Parent Pointers with Top Down Tree Traversals.** In order to evict a page, *LeanStore* has to unswizzle it by replacing the virtual memory pointer to the page in its parent node with the page identifier. This means we need to be able to get the parent node of each node we want to evict. The initial *LeanStore* implementation [Le18] achieves this by storing a parent pointer in each buffer frame. Although this seems an efficient way to reach the parent, it imposes several restrictions and complicates the system. Concretely, whenever we split a B-tree node, we have to update the parent pointer in the buffer frames of half of its child nodes. Also, care needs to be taken when latching the parent, as in general lock coupling in B-trees is based on the invariant that latches may only be acquired top-down. In our current design, we refrain from storing parent pointers and use a generic *findParent* method that returns a handler to the parent by traversing the data structure via its usual access path from the root down to the requested node. The same mechanism is also used when splitting or merging a node in the B-tree, as these also require access to the parent node.

### 6.3 Scaling to Multiple NVMe SSDs

For efficiency and correctness reasons, the system must synchronize I/O requests to prevent different threads from loading the same page into possibly different locations in the buffer pool simultaneously. To this end, we use an I/O stage to explicitly serialize I/O requests by page identifier (PID) like traditional buffer managers. The I/O stage uses a hash table that maps each PID to its state and a waiting queue that other threads sleep in until the page I/O finishes. With one SSD, the single I/O stage with the global mutex in the original *LeanStore* design from 2018 was enough to saturate the SSD bandwidth. With today’s PCIe 4.0, we can attach an array of, e.g., ten NVMe SSDs [HHL20], where each drive can deliver up to 7000 MB/s read bandwidth and 3800 MB/s write bandwidth. In such a system, we need high concurrency to issue a large number of parallel I/O requests to fully utilize all SSDs. The

global mutex that protects the I/O hash table<sup>5</sup> would quickly become a scalability bottleneck and prevent us from reaching exploiting the maximum bandwidth that modern flash drives can deliver.

**Partitioned I/O Stage.** We partition our I/O stage according to the page identifier (PID) as shown in Fig. 4. The least significant bits of a Page ID determines the I/O stage partition that we will use to handle the I/O operation. Page IDs are assigned randomly and as a result, the I/O operations are evenly distributed among the different partitions regardless of the workload characteristics. This reduces the contention on the mutex that protects the I/O stage significantly and allows more worker threads to read pages in parallel. The number of partitions is a freely configurable parameter. As a heuristic, we set the number of I/O partitions at least equal to the number of threads in the system to maximize bandwidth. Note that the buffer pool is still not partitioned and a buffer frame can host a page that is assigned to any of the I/O partitions.

**Background Page Provider Threads.** In the original design, foreground worker threads cool and evict the pages. This adds latency to transactions that suddenly get paused to evict the needed free pages. We deviate from this design and implement a page provider thread routine that cool and evict pages in the background [Ha20]. To fully utilize the available IOPS, multiple page provider threads can be launched in parallel. Moreover, to write back dirty pages, each thread submits *batches* of write commands to the kernel using a native asynchronous interface like `libaio` or `io_uring`. As we have removed the cooling stage, synchronizing the background threads become simpler.

## 7 Logging

In our first logging paper [Ha20], we describe a scalable distributed logging scheme that uses persistent memory (pmem) as a first landing stage for WAL records. However, the slow adoption of pmem and its total abandonment by Intel encouraged us to implement a logging scheme that is optimized for flash SSD characteristics solely. LeanStore now also implements Early Lock Release (ELR) [De84; Jo10] which reduces transaction abort rates by releasing write locks without having to wait for flushing the log. Both ELR and distributed logging amplify the cost of cross-worker logging dependencies. Therefore, we introduce a new fine-granular dependency tracking mechanism to reduce the false positive dependencies. In the following, we assume that transactions run in Read Committed or Snapshot Isolation level.

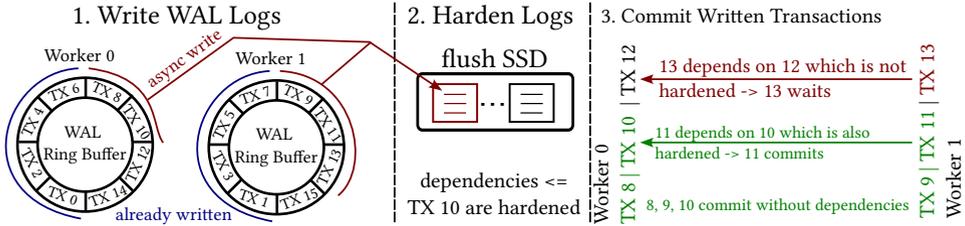


Fig. 5: Group Commit: Each round collects and writes WAL records on SSD before calling fsync(). Consequently, it signals all safe-to-commit transactions if their dependencies are also hardened

### 7.1 Group Commit

Despite being a widely implemented technique in database systems, few research papers describe SSD-optimized group commit. In Fig. 5, we illustrate our group commit implementation. Group commit works in rounds of 3 steps each. Most of the work is done by a background Group Committer Thread (GCT). Worker threads only push the *pre-committed* transactions, which have already passed the concurrency control validation phase, into a ready-to-commit queue and the GCT takes on from this point. The first step in a group commit round is to write WAL records from every worker thread on SSD. On Linux, we use the asynchronous IO interface from libaio to batch all log writes and submit them using a single system call. Once the writes are done, we flush the block device with *fsync* to make sure that the log records we have just written are durable. Consequently, we calculate the new safe set of transactions that are hardened and ready to signal their commit to the client. With this information in hand, we can commit the *pre-committed* transactions in each worker that have their own log and their dependencies hardened.

### 7.2 Dependency Tracking

With distributed logging [Ha20] and Early Lock Release (ELR), transaction commit dependencies to other transactions must be determined. With ELR, a transaction T1 releases the write locks in its pre-commit phase before having its WAL records hardened on SSD. This means that a transaction T2 that reads from T1 cannot commit before T1. This dependency must be determined and hardened with the WAL commit log to preserve this dependency at recovery. Dependencies across transactions processed by the same worker thread have no impact on performance as they are processed serially. Cross-worker thread dependencies, on the other hand, impact latencies negatively as the system has to wait until all of the transaction’s dependencies are hardened. Coarse-grained dependency tracking using the Global Sequence Number (GSN) on each page leads to many false positive dependencies.

<sup>5</sup> The I/O hash table tracks in-flight I/O operations and is required to avoid race conditions which could otherwise occur when the same uncached page is accessed multiple times [Le18].

For example, if T1 and T2 concurrently update different tuples on the same page, GSN synchronization will lead to a false dependency between the two transactions.

**System vs. User Transactions** Before we define the dependency rule, let us mention the two types of transactions in our storage engine and how they differ from each other. *User TXs* execute the commands of the storage engine client. They have a start and commit timestamp that are drawn from the same global atomic counter. Their side effects are only visible after commit and only to user transactions that started from the commit. *System TXs* are triggered by the storage engine itself [Gr11]. For example, they are used to split or merge B-tree pages, manipulate the free space inventory and their side effects are immediately visible to all other transactions. They have do not change the data written by the user but only impact the physical representation. A system transaction exclusively locks all the pages it wants to modify before applying the changes. The start and commit timestamp of a system TX are thus always equal and drawn from another global atomic counter separate to the user transactions.

**Commit Condition.** Because read committed is the lowest sensible isolation level, user transaction dependencies can be tracked at transaction granularity instead of log sequence numbers like LSNs or GSNs. A user transaction depends on previous user and system transactions. For system transactions, it is enough to track the maximum system transaction ID it has witnessed while reading pages. Each page stores the maximum system transaction that had written on it. For the user one, we can either use the user TX start timestamp in case of snapshot isolation level or the start timestamp for the most-recent (pre-)committed user transaction it has read from. The Group Commit Thread calculates the two watermarks: `user_tx_hardened_up_to` and `system_tx_hardened_up_to` of all workers during each round and commits the pre-committed transactions that are below them.

## 8 Evaluation

In this section, we first experimentally compare LeanStore with WiredTiger and RocksDB using TPC-C and YCSB. Both benchmarks do not involve complex analytical operations like join or aggregation and are fully supported in all of the competing transactional key-value storage engines. We then show the effectiveness of the B-tree optimizations described in Sect. 4 and of the commit dependency tracking discussed in Sect. 7. All benchmarks are implemented as a C++ client linked with the storage engine library. Unless explicitly stated, all engines have been configured to run under snapshot isolation. WiredTiger supports the lower isolation mode read uncommitted (RU) only for read-only transactions which we denote as WiredTiger (RU) and evaluate to measure the impact of snapshot construction. All experiments were performed on a single-socket server with an AMD EPYC 7713 64-Core CPU (128 hardware threads) with 512 GB of DRAM running Linux. For storage, we use a RAID-0 of ten 3.8 TB Samsung PM1733 SSDs using XFS as file system.

## 8.1 In-Memory Scalability

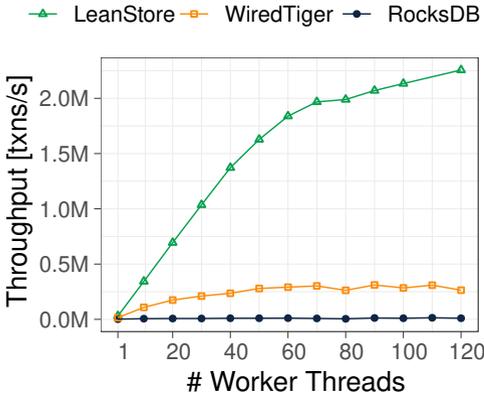


Fig. 6: TPC-C in-memory scalability

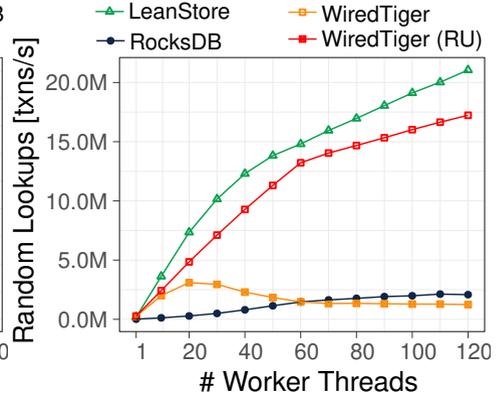


Fig. 7: YCSB read-only in-memory scalability

Let us first investigate in-memory scalability. We compare the performance of LeanStore with its competitors on TPC-C (Fig. 6) (with warehouse affinity) and a YCSB-like random lookup workload (Fig. 7). In both experiments, the buffer pool is configured to be large enough to fit the data sets.

On TPC-C, we see that LeanStore performs much better than the other two systems at higher thread counts. Note that, as expected, when more than 64 worker threads are used, the scalability is slightly worse due to hyperthreading. The absolute performance of over 2 million TPC-C transactions per second shows that LeanStore can compete with the fastest in-memory systems (despite also supporting out-of-memory workloads).

On the read-only random lookup workload, only WiredTiger under the lower isolation mode read uncommitted (RU) comes close to LeanStore snapshot isolation scalability. Despite running on higher isolation level, LeanStore remains around 30% faster. In the absence of concurrency control (RU), hazard pointers and free-lock techniques employed in WiredTiger make the index lookup scales similar to LeanStore’s pointer swizzling and optimistic lock coupling. This result highlights the importance of scalability primitives on modern hardware. In WiredTiger, the cost of constructing a snapshot is proportional to the number of active threads in the systems. At higher threads count, snapshot construction costs more the actual lookup which leads to the performance deterioration beyond 20 threads. With our novel’s MVCC implementation [AL23], LeanStore scales linearly without hard coding any optimization for single-statement or read-only transactions.

## 8.2 Out-Of-Memory Scalability

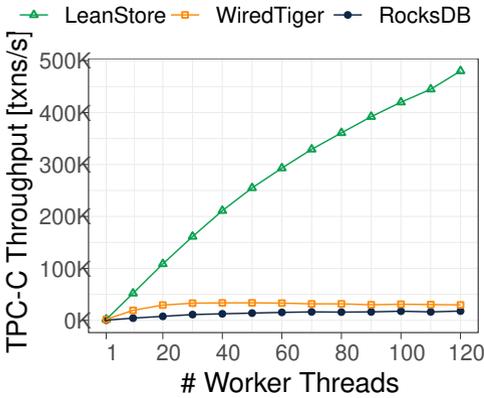


Fig. 8: TPC-C out-of-memory scalability

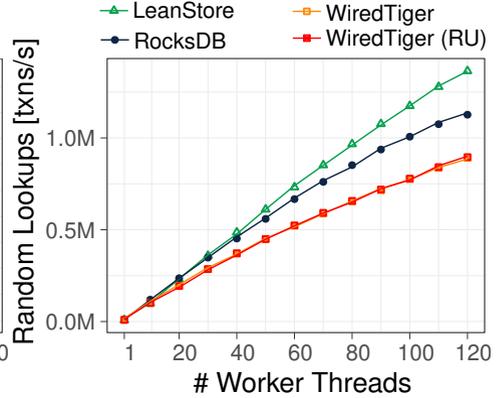


Fig. 9: YCSB read-only out-of-memory scalability

For the next two experiments we configured the buffer pool to be smaller than the data set. For TPC-C we use a 10 GB buffer pool for a 24 GB data set, and for YCSB we use a 10 GB buffer pool for a 100 GB data set. Fig. 8 shows that only LeanStore scales well, reaching 500 thousand TPC-C transactions per second in an out-of-memory setting. Interestingly, on the read-only YCSB benchmark shown in Fig. 9, the gap to the other systems is smaller. The impact of snapshot construction in WiredTiger diminishes because of the dominance of I/O cost. Apparently, these systems can handle read-only workloads reasonably well, but not write-intensive workloads like TPC-C. The observed performance with 120 threads of over 1 million random reads per seconds can be explained by being entirely I/O latency bound: the latency of a single read I/O is about 100 microseconds [HHL20], which corresponds to 10K per second per thread. Our hardware setup could theoretically support an order of magnitude higher I/O rates, but this would require more concurrent operations.

## 8.3 B-tree Optimizations

Tab. 2: Performance impact of B-Tree optimizations on single-threaded lookup performance in a B-tree of 10 million 8-byte integers or 6.4 million strings with an average length of 63 bytes

|          | Integers     |            |             | Strings      |           |             |
|----------|--------------|------------|-------------|--------------|-----------|-------------|
|          | operations/s | instr./op. | L1-miss/op. | operations/s | intr./op. | L1-miss/op. |
| baseline | 1,094,092    | 1,255      | 71          | 914,693      | 1,266     | 99          |
| + prefix | 1,127,396    | 1,261      | 72          | 1,013,265    | 1,252     | 83          |
| + heads  | 1,811,594    | 590        | 36          | 1,296,897    | 1,067     | 58          |
| + hints  | 2,427,184    | 567        | 17          | 1,383,918    | 1,137     | 45          |

Let us next evaluate the impact of the B-tree performance optimizations. We look at the single-threaded in-memory lookup performance. As keys, we use 10 million dense random 8-byte integers (plus 8-byte payload), and 6.4 million real-world URLs with an average length of 63 bytes. Tab. 2 shows the performance, CPU instructions and L1 cache misses for both workloads. The baseline is the basic slotted page layout without any optimizations. We then cumulatively enable the prefix, heads, and hints optimizations. As the table shows, each optimization improved performance significantly for both workloads. What is interesting is that in the baseline case, the integer workload is only 20% faster than the string workload, while with all optimizations the gap increases to 76%. This is because for string data set and its long keys (63 bytes), many of the cache misses are hard to avoid. In the integer case, on the other hand, the optimizations are extremely effective, reducing the number of L1 misses by from 71 to 17 and improving overall performance by 2.2 $\times$ .

#### 8.4 Commit Dependency Tracking

Tab. 3: Percentage of cross thread transaction dependencies (lower is better)

| Tracking Granularity | Warehouse Affinity | Cross Warehouse |
|----------------------|--------------------|-----------------|
| Page Wise            | 52.0%              | 95.5%           |
| Tuple Wise           | 3.9%               | 91.9%           |

In our final experiment, we demonstrate the effectiveness of Commit Dependency Tracking by measuring the ratio of remote flushes necessary. We use TPC-C benchmark with 120 worker threads and 120 warehouses, which corresponds to about 15GB of data. In [Ha20] we proposed a lightweight page wise tracking scheme, which we now compare with tuple-wise tracking described in Sect. 7. Tab. 3 shows that tuple-wise tracking reduces the number of log flushes in both TPC-C settings. In the “Cross Warehouse” setting, many conflicts are unavoidable, but even here the tuple-wise scheme is more precise. With “Warehouse Affinity”, there are few logical conflicts, and tuple-wise tracking reduces the flush rate from 52% to 4%.

## 9 Summary

The goal of LeanStore is to build a high-performance storage engine optimized for multi-core CPUs and NVMe SSDs. In this paper, we describe some important LeanStore components for the first time. Seemingly intricate implementation details of the B-tree, synchronization, buffer management, and logging are crucial for overall performance, scalability, and code maintainability. While the goal of LeanStore has stayed the same since the start of the project, many internals have changed and we expect this evolution to continue. For example, we have recently designed an OS-assisted buffer manager [Le23], which we are now considering for LeanStore.

**Acknowledgments.** This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 447457559.

## References

- [AL21] Alhomssi, A.; Leis, V.: Contention and Space Management in B-Trees. In: CIDR. 2021.
- [AL23] Alhomssi, A.; Leis, V.: Scalable and Robust Snapshot Isolation for High-Performance Storage Engines. In: Under Submission. 2023.
- [An19] Antonopoulos, P.; Budovski, A.; Diaconu, C.; Saenz, A. H.; Hu, J.; Kodavalla, H.; Kossmann, D.; Lingam, S.; Minhas, U. F.; Prakash, N.; Purohit, V.; Qu, H.; Ravella, C. S.; Reisteter, K.; Shrotri, S.; Tang, D.; Wakade, V.: Socrates: The New SQL Server in the Cloud. In: SIGMOD. 2019.
- [AW22a] AWS: Amazon EC2 I3en Instances, <https://aws.amazon.com/ec2/instance-types/i3en/>, 2022.
- [AW22b] AWS: Lsv2-series, <https://learn.microsoft.com/en-us/azure/virtual-machines/lsv2-series>, 2022.
- [Bi22] Binna, R.; Zangerle, E.; Pichl, M.; Specht, G.; Leis, V.: Height Optimized Tries. *ACM Trans. Database Syst.* 47/1, 3:1–3:46, 2022.
- [Bö20] Böttcher, J.; Leis, V.; Giceva, J.; Neumann, T.; Kemper, A.: Scalable and robust latches for database systems. In: DaMoN. 2020.
- [BU77] Bayer, R.; Unterauer, K.: Prefix B-Trees. *ACM Trans. Database Syst.* 2/1, pp. 11–26, 1977.
- [De84] DeWitt, D. J.; Katz, R. H.; Olken, F.; Shapiro, L. D.; Stonebraker, M.; Wood, D. A.: Implementation Techniques for Main Memory Database Systems. In: SIGMOD. 1984.
- [Di13] Diaconu, C.; Freedman, C.; Ismert, E.; Larson, P.; Mittal, P.; Stonecipher, R.; Verma, N.; Zwilling, M.: Hekaton: SQL server’s memory-optimized OLTP engine. In: SIGMOD. Pp. 1243–1254, 2013.
- [Do21] Dong, S.; Kryczka, A.; Jin, Y.; Stumm, M.: RocksDB: Evolution of Development Priorities in a Key-value Store Serving Large-scale Applications. *ACM Trans. Storage/*, 2021.
- [Fä11] Färber, F.; Cha, S. K.; Primsch, J.; Bornhövd, C.; Sigg, S.; Lehner, W.: SAP HANA database: data management for modern business applications. *SIGMOD Rec.* 40/4, pp. 45–51, 2011.
- [GL01] Graefe, G.; Larson, P.: B-Tree Indexes and CPU Caches. In: ICDE. 2001.
- [Gr04] Graefe, G.: Write-Optimized B-Trees. In: VLDB. Pp. 672–683, 2004.
- [Gr11] Graefe, G.: Modern B-Tree Techniques. *Found. Trends Databases* 3/4, pp. 203–402, 2011.
- [Gr14] Graefe, G.; Volos, H.; Kimura, H.; Kuno, H. A.; Tucek, J.; Lillibridge, M.; Veitch, A. C.: In-Memory Performance for Big Data. *PVLDB* 8/1, pp. 37–48, 2014.

- [Ha20] Haubenschild, M.; Sauer, C.; Neumann, T.; Leis, V.: Rethinking Logging, Checkpoints, and Recovery for High-Performance Storage Engines. In: SIGMOD. Pp. 877–892, 2020.
- [HHL20] Haas, G.; Haubenschild, M.; Leis, V.: Exploiting Directly-Attached NVMe Arrays in DBMS. In: CIDR. 2020.
- [Jo10] Johnson, R.; Pandis, I.; Stoica, R.; Athanassoulis, M.; Ailamaki, A.: Aether: A Scalable Approach to Logging. PVLDB 3/1, pp. 681–692, 2010.
- [Ke12] Kemper, A.; Neumann, T.; Funke, F.; Leis, V.; Mühe, H.: HyPer: Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing. IEEE Data Eng. Bull. 35/1, pp. 46–51, 2012.
- [Le16] Leis, V.; Scheibner, F.; Kemper, A.; Neumann, T.: The ART of practical synchronization. In: DaMoN. 2016.
- [Le18] Leis, V.; Haubenschild, M.; Kemper, A.; Neumann, T.: LeanStore: In-Memory Data Management beyond Main Memory. In: ICDE. Pp. 185–196, 2018.
- [Le23] Leis, V.; Alhomssi, A.; Ziegler, T.; Loeck, Y.; Dietrich, C.: Virtual-Memory Assisted Buffer Management. In: SIGMOD. 2023.
- [LHN19] Leis, V.; Haubenschild, M.; Neumann, T.: Optimistic Lock Coupling: A Scalable and Efficient General-Purpose Synchronization Method. IEEE Data Eng. Bull. 42/1, pp. 73–84, 2019.
- [LKN13] Leis, V.; Kemper, A.; Neumann, T.: The adaptive radix tree: ARTful indexing for main-memory databases. In: ICDE. Pp. 38–49, 2013.
- [Lo01] Lomet, D. B.: The Evolution of Effective B-tree: Page Organization and Techniques: A Personal Account. SIGMOD Rec. 30/3, pp. 64–69, 2001.
- [Mo22] MongoDB: WiredTiger Storage Engine, <https://docs.mongodb.com/manual/core/wiredtiger/>, 2022.
- [NF20] Neumann, T.; Freitag, M. J.: Umbra: A Disk-Based System with In-Memory Performance. In: CIDR. 2020.
- [Sa22a] Samsung: PCIe Gen 4-enabled PM1733 SSD, <https://semiconductor.samsung.com/ssd/enterprise-ssd/pm1733-pm1735/mzwlj3t8hbls-00007/>, 2022.
- [Sa22b] Samsung: PM1743, <https://semiconductor.samsung.com/ssd/enterprise-ssd/pm1743/>, 2022.
- [SW13] Stonebraker, M.; Weisberg, A.: The VoltDB Main Memory DBMS. IEEE Data Eng. Bull. 36/2, pp. 21–27, 2013.
- [Tu13] Tu, S.; Zheng, W.; Kohler, E.; Liskov, B.; Madden, S.: Speedy transactions in multicore in-memory databases. In: SIGOPS. Pp. 18–32, 2013.

- [Ve17] Verbitski, A.; Gupta, A.; Saha, D.; Brahmadesam, M.; Gupta, K.; Mittal, R.; Krishnamurthy, S.; Maurice, S.; Kharatishvili, T.; Bao, X.: Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In: SIGMOD. 2017.
- [Wa18] Wang, Z.; Pavlo, A.; Lim, H.; Leis, V.; Zhang, H.; Kaminsky, M.; Andersen, D. G.: Building a Bw-Tree Takes More Than Just Buzz Words. In: SIGMOD Conference. ACM, pp. 473–488, 2018.



# PostBOUND: PostgreSQL with Upper Bound SPJ Query Optimization

Rico Bergmann<sup>1</sup>, Axel Hertzschuch<sup>1</sup>, Claudio Hartmann<sup>1</sup>, Dirk Habich<sup>1</sup>, Wolfgang Lehner<sup>1</sup>



**Abstract:** A variety of query optimization papers have shown the disastrous effect of poor cardinality estimates on the overall runtime for arbitrary select-project-join (SPJ) queries. Especially, underestimating join cardinalities for multi-joins can lead to catastrophic join orderings. A promising solution to overcome this problem is query optimization based on upper bounds for the join cardinalities. In this domain, our proposed UES concept is presently the most efficient technique featuring a simple, yet effective upper bound for an arbitrary number of joins. To foster research in that direction, we introduce *PostBOUND*, our generalized framework to seamlessly integrate upper bound SPJ query optimization in PostgreSQL. *PostBOUND* provides abstractions to calculate arbitrary upper bounds, to model joins required by an SPJ query and to iteratively construct an optimized join order. To highlight the extensibility of *PostBOUND*, and to show the research potential, we additionally present two tighter upper bound UES variants using top-k statistics in this paper. In our evaluation, we show the efficiency and applicability of *PostBOUND* on different workloads as well as using different PostgreSQL versions. Additionally, we evaluate both presented tighter upper bound variant ideas.

**Keywords:** SPJ queries; join order; join cardinalities; upper bound; generalization

## 1 Introduction

The optimization of arbitrary select-project-join (SPJ) queries is still an open research topic and far from being solved [Le15]. For example, one of the most challenging and open issues for complex SPJ queries is finding a good join order [CBS19, He21, Le15]. To tackle this issue, the majority of existing approaches requires reliable precise cardinality estimates for arbitrary joins including joins over intermediate join results and pre-filtered base tables. To provide these reliable estimates, traditional techniques frequently rely on basic heuristics that may assume predicate independence and a uniform distribution of attribute values [Le15]. However, relying on these assumptions can lead to disastrous join orderings [Le15]. Thus, various sophisticated techniques for the join cardinality estimation have been proposed in recent years. On the one hand, sampling approaches seem appealing [Le17, MH20, Zh18], but they do not scale well to many joins [CY17, Zh18]. On the other hand, modern estimation approaches rely on machine learning techniques [Ki19a, Wo19] as they are able to model complex data characteristics. However, these ML approaches do not yet cover all relevant filter predicate types and their training depends on executing a plethora of joins, which may take days or even weeks [Wo20, Wo21].

<sup>1</sup> Technische Universität Dresden, Database Research Group, 01062 Dresden, Germany,  
{rico.bergmann1,axel.hertzschuch,claudio.hartmann,dirk.habich,wolfgang.lehner}@tu-dresden.de

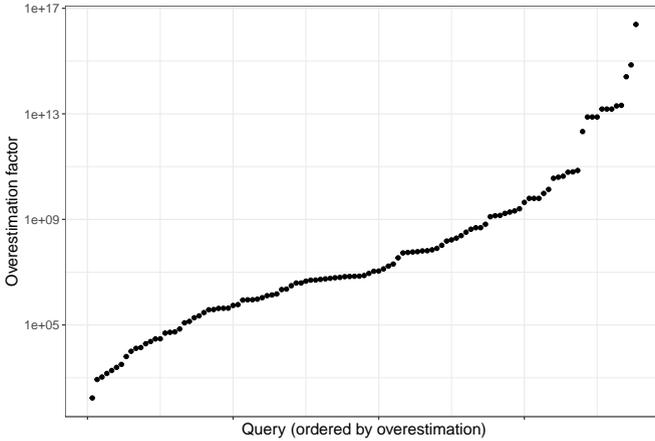


Fig. 1: Upper bound overestimation of UES for the Join-Order-Benchmark.

Thus, state-of-the-art approaches to find good join orders based on reliable cardinality estimates do not seem to be the right way. In contrast to that, approaches using guaranteed upper bounds for join cardinalities are a very promising alternative leading to better and more robust join ordering for complex SPJ queries [CBS19, De22, He21]. In that direction, we have recently introduced a novel concept called *UES* [He21]. The most outstanding feature of UES is its simplicity, achieved by three building blocks:

*U-Block*: Assuming only basic attribute statistics and accurate selectivity estimates for filters over base tables, we defined a simple, yet effective Upper bound for an arbitrary number of joins. In particular, our UES upper bound calculates the worst-case cardinality using only the maximum value frequency per join attribute.

*E-Block*: Appropriately Enumerating joins according to our upper bound effectively prevents overly aggressive (sometimes disastrous) join orderings.

*S-Block*: To guarantee accurate selectivity estimates even for complex filters in SPJ-queries required by the U-Block, we propose using customized Sampling strategies. In case of less challenging filter operations, other synopsis such as standard histograms might be used.

**Our Contributions:** In this paper, we introduce *PostBOUND*, our generalized framework implementation around the general UES concept to seamlessly integrate upper bound SPJ query optimization in PostgreSQL. *PostBOUND* provides abstractions to integrate arbitrary upper bounds for join cardinalities, to model joins required by a query and to determine an optimized join order. These abstractions are supplemented by a customized version of the UES algorithm and accompanied by other assisting components such as estimation strategies for base table filters or an infrastructure for physical operator selection. Our overall

framework is implemented in Python with extensibility being one of the primary design goals.

To demonstrate this extensibility and to show ongoing research potential, we additionally introduce two tighter upper bound variant ideas as a generalization of UES. Fig. 1 exemplarily illustrates the overestimation factor of the UES upper bound compared to the real query results for the *Join-Order-Benchmark (JOB)* [Le15]. It is clearly visible that the overestimation in the range from  $10^3$  to  $10^{16}$  is extreme, leading to a very pessimistic approach. One of the problems of such a high overestimation is that the determined join order for a SPJ query could be too defensive and there could be a join order providing a faster runtime. Another disadvantage is that the upper bound cannot be used for the physical operator selection [He21, He22]. Thus, an obvious optimization opportunity of our UES approach is to compute tighter upper bounds to overcome these disadvantages.

**Contributions in Detail and Outline:** To summarize, we make the following contributions, which also define the outline of this paper:

- In Section 2, we recap our UES concept as foundation for the remainder of the paper.
- Our developed PostgreSQL extension called *PostBOUND*<sup>2</sup> is described in Section 3. *PostBOUND* enables seamless integration of upper bound SPJ query optimization in PostgreSQL and *PostBOUND* is designed as an extensible framework.
- We enhance the UES concept with an idea for a generalized approach to tighten our upper bound based on top-k statistics in Section 4.
- Section 5 presents selective results of our comprehensive evaluation. In particular, we focus on the evaluation (i) of the upper bound optimization on different workloads as well as different PostgreSQL versions and (ii) of the impact of the tighter bounds.

Finally, we close the paper with related work in Section 6 before concluding in Section 7.

## 2 UES - Join Ordering with Simple Upper Bound

In contrast to state-of-the-art approaches (see Section 6), our UES concept [He21] follows a completely different idea to determine good join orderings for SPJ queries: instead of trying to obtain precise estimates for join cardinalities, it calculates theoretical upper bounds of the sizes of intermediate result sets and uses these bounds in a heuristic join enumeration algorithm. This concept is based on the insight that the duration of query workloads is often dominated by the runtime of very few queries that take an exceptionally long amount of time. Meanwhile, most queries in a typical workload can be answered rather quickly. This phenomena is referred to as *tail latency*. Thus, a novel query optimization strategy should focus on improving these long-running queries, instead of speeding up queries that are already fast. However, this can usually not be achieved by improving the runtime in the

<sup>2</sup> *PostBOUND* is available open-source at <https://github.com/rbergm/PostBOUND>. The *btw23-reproducibility* branch is prepared specifically for the reproducibility effort of BTW 2023.

average case, since new outliers can become part of the tail latencies. In contrast, a correct theoretical upper bound can never be wrong in the sense that the true cardinalities exceed the bound. By feeding these worst-case estimates to the join enumeration algorithm, it will probably choose a join order that is too pessimistic in the sense that another join order could have provided a faster runtime. But it will never choose a join order that is too optimistic, i.e. a plan that only works if the intermediate cardinalities are indeed small and takes a much longer time to execute if this hope is not met. Following this general philosophy of focusing on the long running queries, it is acceptable if very fast queries get slightly slower, as long as the tail latencies are removed. In the following, we sketch both the upper bounds for join cardinalities, as well as the heuristic join enumeration algorithm of UES.

Our UES upper bounds of join intermediate cardinalities are estimated via most frequent value statistics on the join columns. In fact, the only metadata necessary to calculate upper bounds with UES are the frequencies of the most common values of the join attributes and the total number of (filtered) tuples per base table. These statistics are then combined with a number of pessimistic assumptions regarding the distribution and correlation of the attribute values. For brevity the calculation is only summarized here, since the final formula is already presented and justified in [He21]. To estimate the size of a join  $R.x \bowtie S.y$  on (filtered) base tables  $R$  and  $S$  while only using the most frequent values for  $R.x$  and  $S.y$ , a first pessimistic assumption is used: the attributes are *assumed* to be uniformly distributed, with the maximum value frequency (MF) therefore also being the only frequency shared by all values. Based on these frequencies and the total number of tuples per (filtered) relation ( $|\sigma(R.x)|, |\sigma(S.y)|$ ), the minimum number of distinct values per attribute can be estimated as  $|\sigma(R)|/MF(R.x)$  for  $R$  and likewise for  $S$ . To combine these per-table values into an estimate for their join result, a second pessimistic assumption is used: the attribute values are *assumed* to overlap perfectly, i.e. each value in  $R.x$  has a matching join partner in  $S.y$  and vice-versa. This leads to  $MF(R.x) \cdot MF(S.y)$  many outgoing tuples per value combination due to the uniformity assumption. Since there are at most  $\min(|\sigma(R.x)|/MF(R.x), |\sigma(S.y)|/MF(S.y))$  many such combinations, the upper bound can be calculated as

$$upper(|\sigma(R) \bowtie \sigma(S)|) := \min\left(\frac{|\sigma(R)|}{MF(R.x)}, \frac{|\sigma(S)|}{MF(S.y)}\right) \cdot MF(R.x) \cdot MF(S.y)$$

Fig. 2 illustrates our UES upper bound concept, using an example. The left-hand side depicts the worst-case – used to derive the upper bound – constrained by the table statistics, while the right-hand side depicts the actual join. Note that  $R$  and  $S$  do not need to be base tables. Instead they can be the result of some other join just as well. For example, without loss of generality, we can assume that  $R = R_1 \bowtie_{R_1.x=R_2.y} R_2$ . In this case,  $|\sigma(R)| = upper(|R_1 \bowtie R_2|)$  and  $MF(R.x) = MF(R_1.x) \cdot MF(R_2.y)$ . This enables the calculation of upper bounds for any join in a recursive manner. One central downside of this approach is the propagation of errors, in this case of overestimated result sizes. Since the estimation of an  $n$ -way join requires estimates of  $n - 1$  smaller joins building on top of each other, estimates grow larger and larger as already shown in Fig. 1.

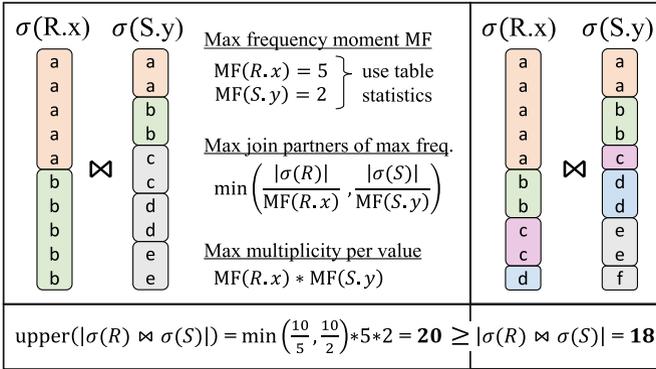


Fig. 2: Illustration of the UES upper bound (taken from [He21]).

Based on this upper bound for join cardinalities, the join order in UES is obtained using a heuristic algorithm [He21]. The UES join enumeration algorithm tries to choose each join such that the sizes of intermediate results are minimized in each step. To make this choice, the upper bounds of candidate joins are calculated and the join with smallest bound is executed. However, this process is only applied to n:m joins. Primary key/foreign key joins (P/K joins) are greedily included as soon as possible. This strategy is justified by a central property of P/K joins: When the foreign key partner is already present in the intermediate result, joining the primary key table may only reduce but never expand the size of the intermediate result. In this sense, P/K joins act as special filters. To facilitate this filtering property even further, a P/K join can be executed as a subquery: suppose an (intermediate) table  $\tilde{T}$  should be joined with a foreign key table  $T_{FK}$ , which in turn has to be joined with a Primary Key table  $T_{PK}$ . The canonical way of executing this join would be  $(\tilde{T} \bowtie T_{FK}) \bowtie T_{PK}$ . The n:m join  $\tilde{T} \bowtie T_{FK}$  would most likely (i.e. following the pessimistic assumption) increase the size of the intermediate result. Afterwards, joining  $T_{PK}$  could potentially reduce the size of the intermediate result again. If such a reduction is guaranteed, executing  $T' := T_{FK} \bowtie T_{PK}$  first and afterwards  $\tilde{T} \bowtie T'$  would minimize work for the second join. Thus, when choosing the next n:m join to execute, UES tries to pre-filter the join partners via P/K joins. If this does not guarantee a smaller intermediate result, the Primary Key partner will be joined after the n:m join has been executed. To some extent, this strategy resembles the well-known pushdown of filter predicates.

To better illustrate the concept of P/K filters, consider part of query 8d of the JOB: `SELECT * FROM cast_info ci, company_name cn, movie_companies mc WHERE cn.country_code = '[us]' AND mc.company_id = cn.id AND ci.movie_id = mc.movie_id`. Overall, this query fragment produces about 59 million result tuples. It contains one P/K join between `movie_companies` and `company_name` and one n:m join between `cast_name` and `movie_companies`. In this case, the P/K join  $mc \bowtie cn$  reduces the cardinality of `movie_companies` from about 5 million to 2 million. This essentially

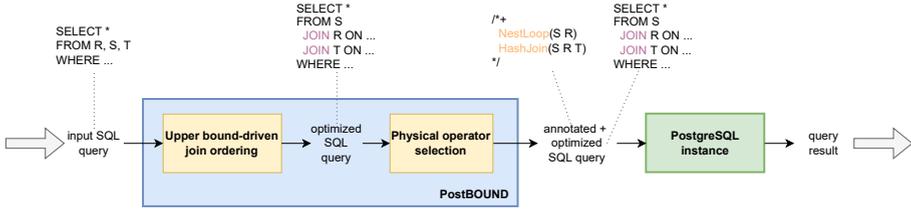


Fig. 3: The basic *PostBOUND* query optimization workflow.

halves the work for the following n:m join between `cast_name` and `movie_companies`, since `company_name` “filtered” its P/K partner.

### 3 PostBOUND - PostgreSQL Extension

In this section, we present *PostBOUND*, our developed framework to seamlessly integrate upper bound-driven optimization of SPJ queries into PostgreSQL. At its core, *PostBOUND* consists of two components as depicted in Fig. 3, which are completely implemented in Python: First, for each incoming SQL query, the join ordering component determines an optimized join order that is used for query execution. The underlying process applies a generalized and extensible implementation of our UES concept (see Section 2). The output of this first component is a rewritten SQL query with an explicit join order. Secondly, our subsequent physical operator selection component enforces the usage of individual physical join and base table scan operators. The resulting query annotations to enforce physical operators are used together with the rewritten SQL of our first component to execute the query with an arbitrary PostgreSQL instance. With this approach, the join order as well as the physical operator selections are fixed by *PostBOUND* using an upper bound-driven optimization concept. Implementation details of both components are described in the following sections<sup>3</sup>.

#### 3.1 Upper Bound-driven Join Ordering Component

The upper bound-driven join ordering is the first component of *PostBOUND* and it focuses on a robust join sequence in order to prevent disastrous execution plans. The main goal of this component is to optimize the join order of an arbitrary SQL query by transforming implicit joins like `SELECT * FROM R, S, T WHERE ...` into an explicit ordering via `JOIN` clauses, such as `SELECT * FROM S JOIN R ON ... JOIN T ON ...`. During query execution, PostgreSQL provides means to enforce such an ordering of `JOIN` statements. Fig. 4 shows the general steps involved in this process: First, a *join graph* is constructed by parsing the

<sup>3</sup> Although *PostBOUND* is currently tailored to PostgreSQL, it can be adapted to other database systems and we plan to integrate different backends in the near future.

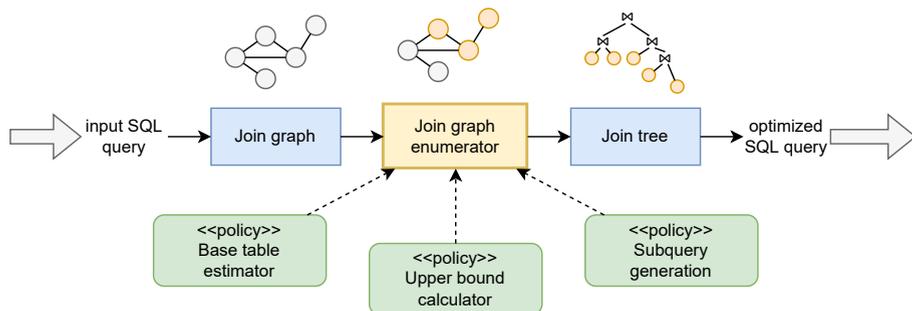


Fig. 4: Interaction between the core *PostBOUND* components for join ordering.

incoming SQL query. This graph serves as the central data structure for optimization and describes the role each table plays in the query. We apply the same distinction between  $n:m$  joined tables and primary key joined tables, as originally proposed in UES [He21]. Based on this graph, the actual optimization loop is executed in the join graph enumerator. During each iteration, the algorithm pulls  $n:m$  candidate tables from the join graph and evaluates their current upper bounds, such that the candidate with minimum bound is inserted into the join tree (tables colored in orange in Fig. 4). This selection focuses only on  $n:m$  joined tables, because primary key/foreign joins are again treated as special filters of the foreign key (and by extension  $n:m$  joined) table that can potentially lower the candidate’s upper bound. Therefore, primary key tables are inserted into the join tree together with their foreign key counterpart. Again, following the UES philosophy, the join between the candidate table and its primary key partners can be executed as a subquery, as the enumerator sees fit (see below for details). Once all tables of the join graph have been included in the join tree, the rewritten query with an explicit join order syntax is constructed based on the derived optimal join order.

In *PostBOUND* we expand on the original UES algorithm in two ways: On the one hand, this entire enumeration process can be adapted with custom policies to modify specific aspects of its behavior. These policies include:

**Base table estimates:** The join ordering component of *PostBOUND* does not require a specific strategy to estimate the number of tuples in a (filtered) base table. It only relies on the existence of a numerical estimate and makes no assumption about how it is obtained. Nevertheless, three basic strategies are already provided and new strategies can be injected easily. The provided strategies include: (i) delegating the estimation process to the PostgreSQL-native optimizer, (ii) sampling a fraction of the filtered table, or (iii) executing the entire filter predicate and counting the result tuples.

**Upper bound calculation and statistics:** To obtain upper bounds of join cardinalities, different strategies have been proposed in recent literature [CBS19, De22, He21]. *PostBOUND* does not restrict the choice of any particular formula, as long as it is capable of the calculation of an upper bound of any  $n$ -ary join. Currently,

the UES formula (see Section 2) and two variations (see Section 4) are provided with *PostBOUND*. Since join upper bound calculation oftentimes relies on specific *statistical information*, each calculation strategy ships its own tailored implementation. The statistics interface receives update information from the optimization loop.

**Subquery generation:** Lastly, the join ordering component of *PostBOUND* also delegates the decision when to generate subqueries for primary key/foreign key joins to custom policies. In this case, four strategies are already supplied by default: (i) a greedy strategy that always generates subqueries, (ii) a defensive strategy that generates subqueries if they guarantee to reduce the size of the foreign key table (as proposed in [He21]), (iii) a “smart” strategy that generates subqueries if a reduction below a certain threshold is guaranteed (which is a generalization of strategy (ii)), and finally, (iv) a strategy that never generates subqueries at all, thereby leaving all join paths linear.

On the other hand, the entire enumeration process is mainly tailored to SPJ queries containing a mixture of n:m and primary key/foreign key joins. To broaden this scope, if an incoming SQL query does not match this structure, it will be handled by specialized procedures:

**Primary key/foreign key queries:** To optimize queries that do not contain any n:m join – such as queries on star- or snowflake schemas – a heuristic approach inspired by UES is used. In this case and since primary key/foreign key joins are bound to never produce more tuples than the cardinality of the foreign key relation, our enumeration algorithm starts with the smallest foreign key table and iteratively includes connected tables according to their respective cardinality estimates. This strategy tries to minimize the number of tuples that have to be processed, but only treats the join order as a local optimization problem. An extension that also considers the join cardinalities could be a natural and effective improvement in future work.

**Cross product queries:** Queries with cross products are characterized by tables that are neither directly nor indirectly linked with join predicates. Using the join graph, this situation can be easily detected through the existence of multiple graph components. Since each of these components represents a complete join graph on its own, an optimized upper bound-driven join order can be obtained per partition. Afterwards, a final join order can be constructed by sorting the individual join trees according to their upper bounds.

**Composite join predicates:** In contrast to the two previous extensions, composite join predicates do not influence the join order itself. Rather, composite join predicates need to be handled during the upper bound calculation and are, thus, subject to the policies. However, they still constitute a special case that needs to be considered to ensure independence from specific workloads. Therefore, they are quickly discussed here. For upper bound-driven calculation, composite join predicates can be handled quite naturally: Since a conjunctive predicate requires each of the individual base predicates to be fulfilled, the final upper bound is constrained by the smallest estimate of the base predicates. Thus, the upper bound approach can simply calculate the

minimum of base estimates. Other upper bound approaches may rely on different strategies such as the calculation of mean bounds, but for UES as well as the two presented variants in Section 4, the minimum strategy is used.

Based on these extensions, *PostBOUND* is currently capable of optimizing SPJ queries, as well as some non-SPJ queries, as long as the following requirements are met: (i) each part of the `SELECT` clause is either directly derived from a base attribute or an aggregation of such attributes, (ii) all joins are either equality predicates over base table attributes, or conjunctions of such predicates, (iii) each base table can optionally be filtered using arbitrary predicates that are not optimized further, and (iv) each query can optionally contain a `GROUP BY`, `ORDER BY`, or `HAVING` clause, which are also ignored during optimization. These restrictions are mostly due to technical reasons to keep the implementation effort manageable, rather than being caused by limitations of the underlying ideas.

### 3.2 Physical operator selection component

For an efficient query execution, not only the join order is important, but also the selection of the best-fitting physical operators [He22]. In *PostBOUND*, this selection process is handled by a dedicated physical operator selection component (cf. Fig. 3). Although many state-of-the-art query optimizers intertwine the operator selection and the determination of the optimal join order, the upper bound approach makes this difficult due to the overestimation as shown in Fig. 1. Therefore, new, specific approaches are required. In [He22], we have recently presented such a selection approach using a learning-based concept that allows physical operator decisions for arbitrary join paths based on learned query feedback. Other approaches are also conceivable and thus, we decided to provide a specific component in *PostBOUND* to accommodate such selection approaches.

Since forcing the execution of joins or table scans with specific algorithms depends strongly on the concrete database system, *PostBOUND* focuses on two means supported by PostgreSQL. The first strategy uses runtime variables that modify the behavior of the PostgreSQL planner. For example, the `SET enable_nestloop = 'off';` option used by UES disables nested loop joins globally for all following queries in a workload. Secondly, *PostBOUND* also provides interfaces that force individual joins to be executed with specific operators. This feature is based on the `pg_hint_plan`<sup>4</sup> extension that specifies a number of *query hints*. A hint is essentially a comment preceding an SQL query that modifies the execution and optimization behavior of PostgreSQL for that specific query. For example, the hint `/*+ HashJoin(movies actors) */` would enforce the join between `movies` and `actors` to be executed by a Hash join. How these hints are generated is left to user-specific selection strategies. In our ongoing research, we want to generate query hints for joins based on upper bounds, which however requires tighter upper bounds. One approach to infer such bounds is presented in the following section.

<sup>4</sup> [https://github.com/oss-c-db/pg\\_hint\\_plan/](https://github.com/oss-c-db/pg_hint_plan/)

## 4 Towards Tighter Upper Bounds with Top-K Lists

As presented in the previous section, *PostBOUND* is a novel framework for upper bound-driven query optimization of SPJ queries based on a generalized version of the UES concept [He21]. Since the original UES algorithm only calculates upper bounds based on the most frequent attribute value [He21], this upper bound approach highly overestimates the join intermediate cardinalities (cf. Fig. 1), leading to a very defensive approach and refrains from using physical join operators and join orders that work best on small amounts of join tuples. To overcome that challenge and to demonstrate the extensibility of *PostBOUND*, we introduce two tighter upper bound variant ideas in this section.

A natural extension of the UES upper bound is to consider not only the largest attribute value frequency, but the largest  $k$  frequencies along with their corresponding attribute values. Such information is typically stored in *top-k* lists (sometimes also called *most frequent values*), which basically are an ordered sequence  $((v_1, f_1), \dots, (v_k, f_k))$  of attribute values  $v_i$  and their corresponding frequencies (i.e. number of occurrences)  $f_i$ , such that  $v_1$  is the most frequent value,  $v_2$  the second most frequent value, and so on. Such a top-k list can be used for two basic purposes: for all attribute values that are present in the top-k list of both join partners, their frequency can be used directly to calculate the number of outgoing tuples for a join. For all attribute values that are not contained in the top-k list, their maximum frequency can still be derived based on the minimum frequency in the list, i.e. the  $f_k$  frequency: If the frequency were higher, the attribute value would be present in the top-k list in the first place.

Based on these top-k lists, we devise two basic families of algorithms to calculate an upper bound for a join  $R.a = S.b$ : The *first family* divides both  $R.a$  and  $S.b$  into two disjunct sets, one that encompasses all attributes that are contained in the respective top-k list, and one that contains the remaining values. For all values that are in either top-k list, the number of tuples in the join result can be calculated accurately. For all values that are in neither top-k list, a fallback strategy is used. The *second family* calculates the upper bound by simulating a worst-case join scenario. This is achieved by iteratively joining attribute values from the top-k lists, such that the overall cardinality is maximized. This strategy directly adopts the pessimistic nature of UES in that it considers the absolute worst case distributions of attribute values for both  $R.a$  and  $S.b$ .

In the remainder of this section, we present two exemplary algorithms from both families, starting with the *approximate top-k bound* as an instance of the first family, followed by the *cautious top-k bound* from the second family. These formulas are not the only instances of their respective families and they are by no means perfect. Instead, they serve as a starting point for further research. The section concludes with some remarks regarding the update of top-k lists when considering  $n$ -way joins. To keep notation short, we use the following definitions: we calculate an upper bound for a join  $R.a = S.b$  between attributes  $R.a = (a_1, a_2, \dots, a_n)$  and  $S.b = (b_1, b_2, \dots, b_m)$ , where  $a_i$  and  $b_i$  denote the different attribute values in each column. As long as there is no ambiguity, we refer to each attribute

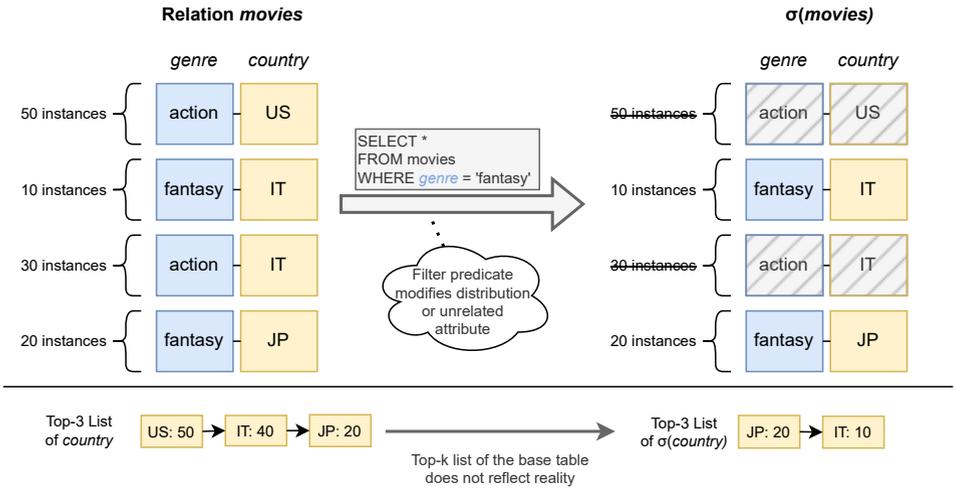


Fig. 5: Static top-k lists can be misleading in the light of filter predicates.

by its relation, e.g. instead of  $R.a$  we simply write  $R$ .  $|R|$  and  $|S|$  denote the number of tuples in each relation. Both attribute sets have an associated top-k list  $\text{top}_R \subseteq R.a$  and  $\text{top}_S \subseteq S.b$ . For each attribute value  $x$ , we define the *attribute value frequency* as follows:

$$AF_R(x) := \begin{cases} |\{a \in R \mid a = x\}| & \text{if } x \in \text{top}_R \\ f_R^* & \text{otherwise} \end{cases}$$

$f_R^*$  denotes the minimum frequency of any value in the top-k list of  $R$ .

#### 4.1 Approximate Top-k Bound

The main idea of the *approximate bound* is to split the calculation into two parts, i.e.  $\text{upper}(R.a = S.b) := \text{upper}_{\text{top}}(R.a = S.b) + \text{upper}_{\text{rem}}(R.a = S.b)$ , where the first bound is directly based on the top-k lists and the second bound accounts for all remaining attribute values with lower frequencies. Deriving an upper bound for values in the top-k lists is pretty straightforward: for each value in either top-k list, its frequency is multiplied by the value frequency in the other top-k list, falling back to  $f^*$  if necessary. Thus,  $\text{upper}_{\text{top}}$  can be expressed as

**Definition 4.1** (Top-k based approximate upper bound).

$$\text{upper}_{\text{top}}(R.a = S.b) := \sum_{a \in R.a} AF_R(a) \cdot AF_S(a) + \sum_{b \in S.b \setminus R.a} AF_R(b) \cdot AF_S(b) \quad (1)$$

Deriving an upper bound for all remaining attribute values is significantly harder: Although in principle the UES formula can be used to estimate the maximum cardinality of two attribute sets based on their maximum frequency, some of these values might have already been processed as part of  $\text{upper}_{\text{top}}$ . Thus, these values should not be considered again in the remaining bound. At the same time, frequencies from the top-k lists cannot simply be used to “initialize” the remaining values, due to a fundamental disconnect between the top-k list of an attribute  $R.a$  and the attribute instances that are actually available during query execution: while top-k lists are computed for all attribute values in a base table, at query runtime, the distribution and count of these attribute value instances may be changed fundamentally after applying filter predicates on the base tables. Consider Fig. 5: the filter predicate completely removes the most frequent value and changes the order of the remaining two attribute values. Since the filter predicate can be entirely independent from the attribute being joined, the top-k list can only be considered as a static upper bound of the true attribute frequencies. Since there is no direct way to determine which attribute instances are actually available after executing the filter predicate as long as only basic statistics are considered, we make the pessimistic assumption that none of the values from the top-k list are actually available and the remaining values lie entirely in the scope of  $\text{upper}_{\text{rem}}$ . Thus, the  $\text{upper}_{\text{rem}}$  bound becomes a full UES bound based on the maximum remaining frequency, i.e.  $f^*$ :

**Definition 4.2** (Remaining UES bound of the approximate top-k bound).

$$\text{upper}_{\text{rem}}(R.a = S.b) := \min\left(\frac{|\sigma(R)|}{f_R^*}, \frac{|\sigma(S)|}{f_S^*}\right) \cdot f_R^* \cdot f_S^* \quad (2)$$

The issue of top-k lists that are unrelated to the attribute instances is actually also present when calculating the  $\text{upper}_{\text{top}}$  bound: the processed frequencies can significantly overestimate the number of tuples that are truly available. However, in this case, we know the total number of available tuples as well as the number of processed tuples per relation. Thus, we can slightly mitigate the impact of overestimation by constructing an adjustment factor for  $R$  as well as for  $S$ . Each factor is simply the ratio between available tuples and processed tuples. The factors will be applied as soon as they are smaller than 1 (i.e. there was an overestimation). Strictly speaking, this trick assumes a uniform distribution of the overestimation, i.e. that each attribute value is overestimated by the same fixed delta. This may drop the upper bound for the top-k list below the actual cardinality in some rare cases. By applying the  $\text{upper}_{\text{rem}}$  bound as defensively as presented in Definition 4.2, this issue is largely mitigated, leaving the bound *effectively* as an upper bound. Still, there may be situations where the overestimation via  $\text{upper}_{\text{rem}}$  is not enough to compensate the underestimation caused by the adjustment factors, hence the name of an *approximate* upper bound.

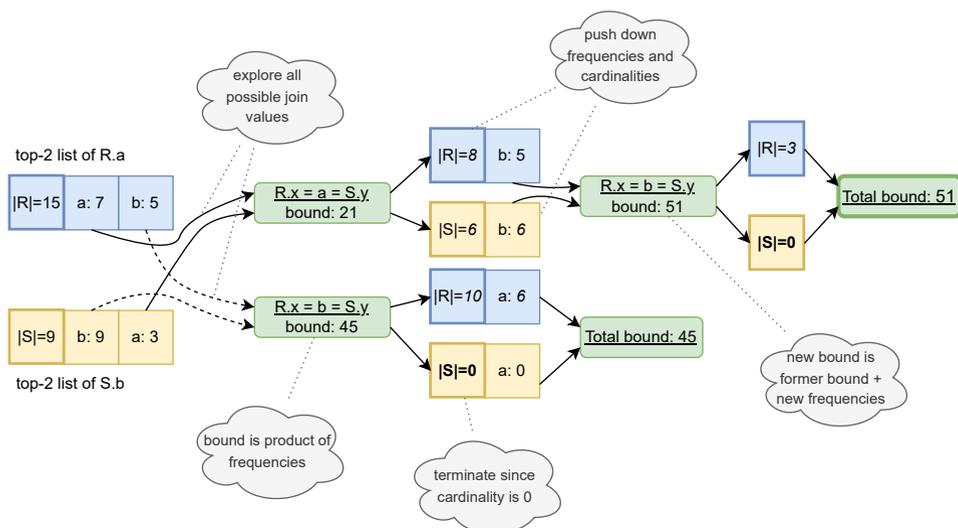


Fig. 6: Join cardinality estimation using the cautious top-k bound.

## 4.2 Cautious Top-k Bound

The approximate nature of the previous bound motivates research in an entirely different direction. The *cautious top-k bound* iteratively tries to construct the maximum number of join tuples based on the entries in the top-k lists and the total number of available tuples. A pseudo-code implementation of this strategy is given in Algorithm 1 and illustrated in Fig. 6 (branches and decisions of the algorithm are depicted in green):

In each iteration, the cautious top-k bound tries to obtain the maximum possible bound given its current state. This is achieved by systematically simulating the number of outgoing tuples for each attribute value in the top-k lists. In Fig. 6, this initially means exploring the attribute values  $a$  and  $b$ . For each value, the size of its partial join result is calculated (line 8) and included in the upper bound. Afterwards, the top-k lists as well as the total number of available tuples are adjusted based on the value that was just “consumed” (lines 9 to 12). To adjust the total number of tuples available for each relation, the value’s frequency is simply subtracted from the current count. Top-k lists are updated to no longer include the candidate value and each frequency including  $f^*$  is enforced to be at most as large as the remaining number of tuples. In Fig. 6, the top-k lists contain just 1 more value after the first selection. At this point, the maximum bound for the smaller top-k lists can be calculated (line 13), leading to a recursive structure. Recursion terminates if either no more tuples are available (line 2), or the top-k lists are empty (line 4). In the first case, both relations have been consumed completely and the current bound is the maximum bound for this branch. In the second case, the remaining values are estimated using the UES bound. Once the bound

---

**Algorithm 1** Pseudo-code implementation of the cautious top-k bound.

---

```

1: function CAUTIOUS_BOUND(top(R), top(S), |R|, |S|, current bound)
2:   if |R| = 0 or |S| = 0 then
3:     return current bound
4:   if top(R) is empty and top(S) is empty then
5:     calculate UES bound based on  $f^*$  and remaining tuples counts
6:     return current bound + UES bound
7:   for all attribute values  $v$  in top(R) and top(S) do
8:      $candidate \leftarrow AF_R(v) \cdot AF_S(v)$ 
9:      $|R'| \leftarrow \max(|R| - AF_R(v), 0)$  ▷ adjust the remaining tuples
10:     $|S'| \leftarrow \max(|S| - AF_S(v), 0)$ 
11:    limit top(R') frequencies and  $f_R^*$  to  $|R'|$  ▷ top(R') := top(R) \setminus v
12:    limit top(S') frequencies and  $f_S^*$  to  $|S'|$  ▷  $f'_i = \min(f_i, |R'|)$ 
13:    value bound  $\leftarrow candidate +$  cautious bound(top(R'), top(S'), |R'|, |S'|, current bound)
14:   return current bound + maximum value bound

```

---

for all candidate values has been determined, the algorithm selects the maximum possible bound and calculates the final bound (line 14).

This strategy effectively determines the upper bound for each permutation of attribute value instances and thus suffers from a large computational complexity if implemented naively. However, a more elaborate implementation can effectively prune large parts of the search space by checking, whether the largest possible bound still contained in the top-k lists can exceed the maximum bound already observed. If this is not the case, there is no point in further exploring the current branch. Furthermore, many bounds of intermediate joins can be re-used if the top-k lists of their recursion branches are equal. For example, if a bound has been calculated for join  $A \bowtie B \bowtie C \bowtie D$  and join  $A \bowtie B \bowtie D \bowtie C$  is explored, the previous results could potentially be re-used. Additional optimization techniques for an efficient implementation should be explored in future research.

### 4.3 Updating Top-k Based Statistics for $n$ -way Joins

Besides the calculation of the upper bounds themselves, the underlying data structures also have to be updated to estimate higher-order joins that involve 3 or more relations. In the case of top-k lists, this issue is twofold: First, the top-k lists of the join attributes have to be combined, integrating the knowledge of both individual lists. This problem can be solved in a straightforward manner: The resulting top-k list contains the sum of both source frequencies for each value in either top-k list, again falling back to the  $f^*$  frequencies as necessary (Fig. 7). The  $f^*$  frequency of the merged top-k list can likewise be estimated as the product of both source  $f^*$  frequencies. The second case is the update of a top-k list that is not directly involved in the join (a *third-party list*). This process is especially important since that list may take part in a later join and the associated frequencies must therefore reflect the current state of the intermediate join result. The fundamental problem for this update is the

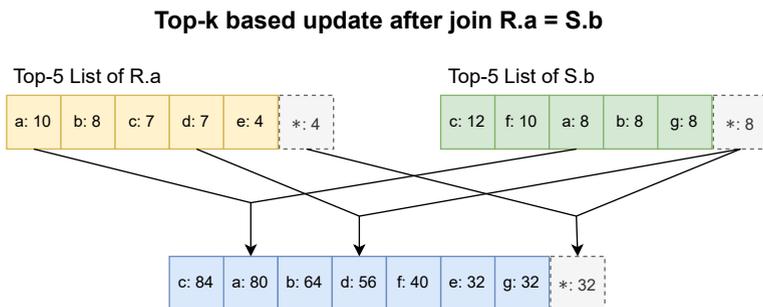


Fig. 7: Merging two top-k lists after a join.

lack of correlation information between the joined top-k list and the third-party list. For an upper bound-driven approach, this demands another pessimistic assumption: In the worst case, each entry in the third-party list could correlate entirely with the maximum frequency of the joined top-k list. Thus, the maximum possible frequency of each updated value in the third-party list is the product of its current frequency and the maximum frequency in the joined top-k list. In general, this strategy will heavily overestimate the frequencies in the updated top-k list, but without additional information or simplifying assumptions it is not possible to figure out how much each frequency has been overestimated. This matches the basic maximum frequency update of UES and generalizes that idea to top-k lists.

## 5 Evaluation

We focus our evaluation on two major aspects of *PostBOUND*: On the one hand, we examine the independence of *PostBOUND* from specific workloads using two well-known benchmarks and different PostgreSQL versions. On the other hand, we explore the potential of upper bound-driven query optimization in more detail. This includes (i) the computed join orders, (ii) the generation of subqueries, (iii) the effect of tighter upper bounds, and (iv) the potential of the selection of physical operators. Generally, all experiments are executed on an Ubuntu 20.04 machine with an Intel i7-6700 HQ processor, 32 GB of main memory and an SSD storage. Unless stated otherwise, we run the benchmarks on PostgreSQL 14.

**Analyzing UES - Join Orders:** To analyze *PostBOUND*'s applicability to different database schemas and workloads, we examine the Join-Order-Benchmark (JOB) [Le15] as well as the Star-Schema-Benchmark (SSB) [Sa16]. Queries of the JOB are optimized with UES using the precise base table estimation and defensive subquery generation strategies (cf. Section 3.1). Precise base table estimation allows for the best reproducibility of the optimized queries, since the estimates are derived directly from the live database. Moreover, the defensive subquery generation was also used in [He21], enabling better comparability with these results. The queries of the SSB are optimized based on the base table estimates of

| Benchmark                   | PostgreSQL v12.4 |        | PostgreSQL v14.2 |        |
|-----------------------------|------------------|--------|------------------|--------|
|                             | Native           | UES    | Native           | UES    |
| Join-Order-Benchmark (JOB)  | 895.8s           | 399.5s | 848.0s           | 377.0s |
| Star-Schema-Benchmark (SSB) | 5.1s             | 5.5s   | 5.1s             | 5.3s   |

Tab. 1: Total runtimes of different benchmarks using different PostgreSQL versions.

the native PostgreSQL optimizer to demonstrate this functionality. The underlying TPC-H database is setup to use a scale factor of 1. In addition to different benchmarks, we also evaluate both workloads on two versions of PostgreSQL: v12.4 was already used in [He21] and v14.2 was the latest release of PostgreSQL at the time initial work on *PostBOUND* began. Table 1 contains the total runtime for each benchmark setting. Native optimization in this context means optimization by the built-in optimizer of PostgreSQL, but without usage of nested loop joins. This constraint matches a similar requirement by UES and enables us to focus on the join order, rather than the operator selection, which is beyond the scope of original UES.

The JOB benchmark results highlight two interesting insights: On the one hand, both PostgreSQL versions show a much better performance of UES compared to the native optimizer. This basically confirms the results presented in [He21]. On the other hand, nearly the entire speedup of UES on both PostgreSQL versions is caused by two queries with catastrophic join orders. In the case of Postgres v14.2, these queries are 8c and 19d. Each of these queries is running more than 200 seconds faster when optimized with UES. Execution of the UES variants takes less than 10 seconds, which demonstrates the disastrous effects bad join orderings can have. However, some queries are also slowed down by UES, although to a much smaller degree. For example, query 7c is slowed down the most by about 4 seconds. Results on SSB are much more similar, which is mostly caused by the much smaller data set. However, the runtimes show that the UES optimized queries are neither significantly better, nor significantly worse than native optimization. Nonetheless, the overall simpler queries of the SSB also stray away from the primary focus of upper bound-driven query optimization, which is intended for complex queries with many joins and complicated filter predicates.

**Analyzing UES - Subquery Generation:** A central idea of UES is the evaluation of primary key/foreign key joins in subqueries to achieve an up-front reduction of the foreign key cardinality. However, introducing subqueries also implies additional pipeline breakers when executing the query in PostgreSQL. Thus, we investigate the impact of subqueries on individual queries next by optimizing the JOB workload with two UES settings: The first one uses the smart subquery generation strategy while the second setting produces entirely linear queries. Based on this setup, for example, query 6c shows the largest speedup of about 33% (5.7s to 3.8s) when using subqueries. The resulting execution plans for both settings are sketched in Fig. 8: executing the join between name and cast\_info as a subquery not only reduces the number of processed tuples, but also allows for a parallel processing of that join. Nevertheless, subqueries do not always lead to a performance benefit. In fact, query 7a is slowed down the most with about 0.6 seconds (13% of the linear runtime). Thus, it is

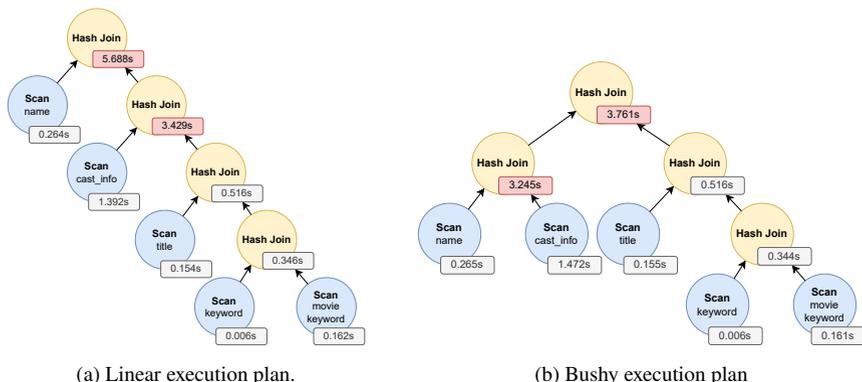


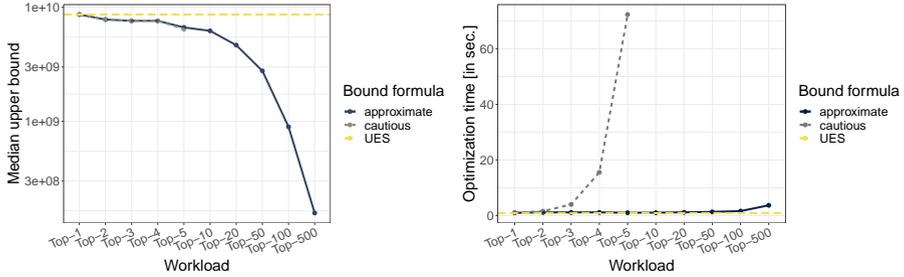
Fig. 8: Generating subqueries for JOB query 6c can improve performance substantially.

important to create subqueries carefully and a smart generation policy – e.g. based on tight upper bounds – seems to work well in these cases.

**Advancing UES - Tighter Upper Bounds:** Tight upper bounds have been proposed as a potential solution for many problems in this paper. In this section, we evaluate how large the impact of the cautious and approximate algorithms presented in Section 4 already is. For this, we vary the length of the top-k lists for both algorithms and compare the results to a UES baseline. In all cases, the JOB queries are upper bound-driven optimized using precise base table estimates and the smart subquery generation policy.

Looking at the median upper bounds across all JOB queries in Fig. 9a indeed reveals a substantial improvement of the bounds as the top-k lists become longer. In fact, using top-500 lists achieves a maximum improvement of factor 210,000 compared to the UES bound. Although this is an extreme case, many upper bounds still improve several orders of magnitude when using top-k lists with just 50 or 100 attribute values. For shorter top-k lists, the improvement is much smaller, as is expected. This is caused by two main factors: shorter lists allow for fewer actual matches of attribute values, causing more usage of the  $f^*$  frequencies. At the same time, shorter lists also allow for less drop-off of frequencies, again resulting in values that are closer to the UES bound.

A central advantage of the approximate formula becomes apparent when looking at the optimization time in Fig. 9b: It stays very low at a maximum of 3 seconds for all queries in the JOB. This is a sharp contrast to the cautious formula, which takes over 1 minute of optimization time already at top-5 lists. This increase in optimization time is also the reason why larger top-k lists are not optimized with the naive implementation of the algorithm. Despite the smaller bounds, only a few join orders are actually updated. In fact, the cautious algorithm only updates 5 queries across all settings. Although this number becomes slightly larger with 31 queries for the approximate formula, it is still quite small considering the 113 queries that are executed. This indicates that UES is sufficient for many queries and



(a) Final bounds of each query. (b) Optimization time of the workloads.

Fig. 9: Impact of the top-k based bound estimation on the JOB queries.

that more elaborate algorithms are required to close in on the estimates of the native optimizer. The few updated queries also lead to very little change of the overall workload runtime. Across all settings, the maximum deviation from UES is about 10 seconds, which is almost negligible considering the complexity of the workload and its overall duration. Even though the evaluated formulas are prototypes by nature, they still achieve a considerable improvement in terms of their upper bounds. This motivates further research in that direction and especially the application of the bounds for different tasks.

**Advancing UES - Physical Operator Selection:** Besides the generation of optimized join orders, *PostBOUND* also enables the selection of physical operators (cf. Section 3.2), thereby removing the restriction to hash joins normally imposed by UES. A natural application of the operator selection is to further exploit the performance gains enabled by subqueries. Since these queries are by design joins between a primary key and a foreign key table, they can be implemented more efficiently using index-nested loop joins. Thus, we again optimize the JOB workload using UES and the smart subquery generation policy. For each resulting subquery, we use *PostBOUND* to generate query hints that enforce the execution of that subquery as an index-nested loop join. Fig. 10 shows the final execution plan for query 8d, which is improved the most by this strategy. Primary key filters are shown in yellow boxes while tables that are n:m joined appear in blue. Each join is annotated by its operator, as well as the point in time when it is executed. The subquery `cast_info`  $\bowtie$  `role_type` is executed as an index-nested loop join in parallel to the join `aka_name`  $\bowtie$  `name`. The choice of operators results in a 45% speedup (5.7s to 2.6s) compared to a pure hash join-based execution. This simple strategy barely scratches the surface of elaborate techniques for physical operator selection, it focuses exclusively on joins between base tables within subqueries and does not consider the upper bounds associated with each join at all. Advanced approaches such as TONIC [He22] could perform much better. Still, the simple strategy of executing all subqueries as index-nested loop joins already shows the potential of appropriate operator choice that can be explored in future work.

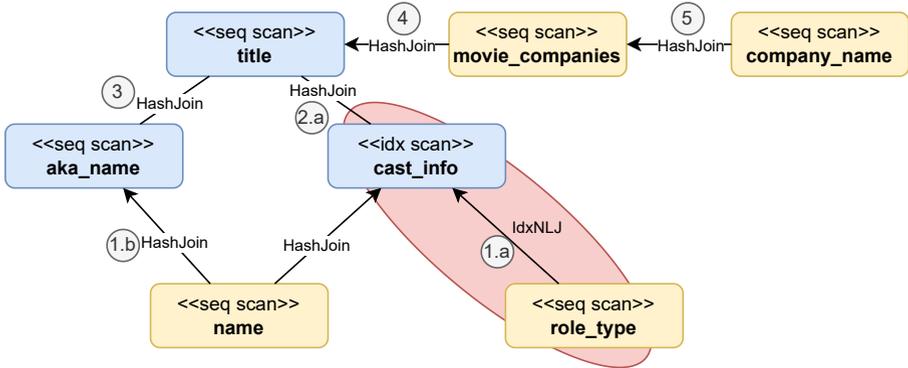


Fig. 10: Join order and operator selection for JOB query 8d. Subqueries use Index-NLJ hints.

## 6 Related Work

Generally, the optimization of SPJ queries entails two major challenges: (i) finding a good join order and (ii) selecting the best-fitting physical join operator for each single join within the chosen join order. According to [Ch98], to solve both challenges, state-of-the-art SPJ query optimizers require precise estimates of intermediate join result sizes (cardinalities). Unfortunately, as shown in [Pe19], ad-hoc estimation techniques are unlikely to achieve such precise estimates. Additionally, Leis et al. [Le15] provide empirical evidence that cost-based optimizers are prone to disastrous planning decisions if precise cardinality estimates cannot be provided. To tackle this issue, recent work investigates more computationally intensive sketches [CBS19, Iz21, Ki19b] or machine learning (ML) approaches [Hi20, Ki19a, Ne21, Wo19, Ya20] to achieve precise cardinality estimates. Beyond cardinality estimation, some ML approaches apply reinforcement learning (RL) for holistic query plan optimization [Kr18, Ma21, Ma19]. For example, Bao [Ma21] learns and injects SQL hints to guide general planning decisions of the underlying optimizer.

An alternative approach to precise cardinality estimates is to compute an upper bound for each intermediate result. This approach originated in the database theory community [GM06]. Atserias et al. [AGM08] introduced a smart formula – nowadays called the AGM bound – that gives a tight upper bound on the query result in terms of the cardinalities of the input tables. This upper bound was improved by the *polymatroid bound*, which takes into account both the cardinalities, and the degree constraints as well as including functional dependencies as a special case [Go12, KNS16, Ng18]. Fundamentally, an upper bound could be used by any cost-based query optimizer in lieu of precise cardinality estimates and this idea was recently pursued by the database systems community, where the upper bound appears under various names such as bound sketch, cardinality bound, or pessimistic cardinality estimator [CBS19, De22, He21]. For example, Cai et al. [CBS19] introduced a

pessimistic cardinality estimator, which uses Count-Min sketches for capturing join crossing correlations. The sketch building process introduces significant overhead when the number of joins increases. In contrast to that, our UES concept [He21] maintains the pessimistic property for cardinality estimation while replacing sketches with a simple formula based on available basic statistics (most frequent attribute values). With *PostBOUND*, we presented a comprehensive framework for all these upper bound approaches in PostgreSQL consisting of two separate components as described in Section 3. Each component focuses on a single challenge for SPJ query optimization, namely finding a good join order and selecting the best-fitting physical operator. Moreover, we introduced and evaluated ideas to improve our simple formula on basic statistics – top-k lists – to tighten the upper bound. In this context, the main challenge is to find upper bound approaches whose computational cost is low.

## 7 Conclusion

The optimization of arbitrary select-project-join (SPJ) queries is still an active research topic. In this context, deriving the necessary cardinality estimates from upper bounds is a promising strategy. To foster research in that direction, we have introduced *PostBOUND*, a generalized framework implementation to seamlessly integrate upper bound SPJ query optimization in PostgreSQL. *PostBOUND* provides abstractions to integrate arbitrary upper bounds, to model joins required by an SPJ query and to iteratively construct an optimized join order. Other than calculating the join order, *PostBOUND* also enables the selection of physical operators, and can thus mimic the entire query optimization process. To highlight the extensibility of *PostBOUND* and to show the research potential, we have additionally presented two tighter upper bound variant ideas using top-k statistics in this paper. Our evaluation has shown the efficiency and broad applicability of *PostBOUND* on different workloads and using different PostgreSQL versions. Moreover, we have also highlighted the impact of the proposed tighter upper bound variants.

## Bibliography

- [AGM08] Atserias, Albert; Grohe, Martin; Marx, Dániel: Size Bounds and Query Plans for Relational Joins. In: FOCS. pp. 739–748, 2008.
- [CBS19] Cai, Walter; Balazinska, Magdalena; Suciu, Dan: Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In: SIGMOD. pp. 18–35, 2019.
- [Ch98] Chaudhuri, Surajit: An Overview of Query Optimization in Relational Systems. In (Mendelzon, Alberto O.; Paredaens, Jan, eds): PODS. pp. 34–43, 1998.
- [CY17] Chen, Yu; Yi, Ke: Two-Level Sampling for Join Size Estimation. In: SIGMOD. pp. 759–774, 2017.
- [De22] Deeds, Kyle; Suciu, Dan; Balazinska, Magda; Cai, Walter: Degree Sequence Bound For Join Cardinality Estimation. CoRR, abs/2201.04166, 2022.

- [GM06] Grohe, Martin; Marx, Dániel: Constraint solving via fractional edge covers. In: SODA. pp. 289–298, 2006.
- [Go12] Gottlob, Georg; Lee, Stephanie Tien; Valiant, Gregory; Valiant, Paul: Size and Treewidth Bounds for Conjunctive Queries. *J. ACM*, 59(3):16:1–16:35, 2012.
- [He21] Hertzschuch, Axel; Hartmann, Claudio; Habich, Dirk; Lehner, Wolfgang: Simplicity Done Right for Join Ordering. In: CIDR. 2021.
- [He22] Hertzschuch, Axel; Hartmann, Claudio; Habich, Dirk; Lehner, Wolfgang: Turbo-Charging SPJ Query Plans with Learned Physical Join Operator Selections. *Proc. VLDB Endow.*, 15(11):2706–2718, 2022.
- [Hi20] Hilprecht, Benjamin; Schmidt, Andreas; Kulesa, Moritz; Molina, Alejandro; Kersting, Kristian; Binnig, Carsten: DeepDB: Learn from Data, not from Queries! *Proc. VLDB Endow.*, 13(7):992–1005, 2020.
- [Iz21] Izenov, Yesdaulet; Datta, Asoke; Rusu, Florin; Shin, Jun Hyung: COMPASS: Online Sketch-based Query Optimization for In-Memory Databases. In: SIGMOD. pp. 804–816, 2021.
- [Ki19a] Kipf, Andreas; Kipf, Thomas; Radke, Bernhard; Leis, Viktor; Boncz, Peter A.; Kemper, Alfons: Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In: CIDR. 2019.
- [Ki19b] Kipf, Andreas; Vorona, Dimitri; Müller, Jonas; Kipf, Thomas; Radke, Bernhard; Leis, Viktor; Boncz, Peter A.; Neumann, Thomas; Kemper, Alfons: Estimating Cardinalities with Deep Sketches. In: SIGMOD. pp. 1937–1940, 2019.
- [KNS16] Khamis, Mahmoud Abo; Ngo, Hung Q.; Suciu, Dan: Computing Join Queries with Functional Dependencies. In: PODS. pp. 327–342, 2016.
- [Kr18] Krishnan, Sanjay; Yang, Zongheng; Goldberg, Ken; Hellerstein, Joseph M.; Stoica, Ion: Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR*, abs/1808.03196, 2018.
- [Le15] Leis, Viktor; Gubichev, Andrey; Mirchev, Atanas; Boncz, Peter A.; Kemper, Alfons; Neumann, Thomas: How Good Are Query Optimizers, Really? *Proc. VLDB Endow.*, 9(3):204–215, 2015.
- [Le17] Leis, Viktor; Radke, Bernhard; Gubichev, Andrey; Kemper, Alfons; Neumann, Thomas: Cardinality Estimation Done Right: Index-Based Join Sampling. In: CIDR. 2017.
- [Ma19] Marcus, Ryan C.; Negi, Parimarjan; Mao, Hongzi; Zhang, Chi; Alizadeh, Mohammad; Kraska, Tim; Papaemmanouil, Olga; Tatbul, Nesime: Neo: A Learned Query Optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, 2019.
- [Ma21] Marcus, Ryan; Negi, Parimarjan; Mao, Hongzi; Tatbul, Nesime; Alizadeh, Mohammad; Kraska, Tim; Bao: Making Learned Query Optimization Practical. In: SIGMOD. pp. 1275–1288, 2021.
- [MH20] Moerkotte, Guido; Hertzschuch, Axel: alpha to omega: the G(r)eek Alphabet of Sampling. In: CIDR. 2020.

- [Ne21] Negi, Parimarjan; Marcus, Ryan C.; Kipf, Andreas; Mao, Hongzi; Tatbul, Nesime; Kraska, Tim; Alizadeh, Mohammad: Flow-Loss: Learning Cardinality Estimates That Matter. Proc. VLDB Endow., 14(11):2019–2032, 2021.
- [Ng18] Ngo, Hung Q.: Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In (den Bussche, Jan Van; Arenas, Marcelo, eds): PODS. pp. 111–124, 2018.
- [Pe19] Perron, Matthew; Shang, Zeyuan; Kraska, Tim; Stonebraker, Michael: How I Learned to Stop Worrying and Love Re-optimization. In: ICDE. pp. 1758–1761, 2019.
- [Sa16] Sanchez, Jimi: A Review of Star Schema Benchmark. CoRR, abs/1606.00295, 2016.
- [Wo19] Woltmann, Lucas; Hartmann, Claudio; Thiele, Maik; Habich, Dirk; Lehner, Wolfgang: Cardinality estimation with local deep learning models. In: aiDM@SIGMOD. pp. 5:1–5:8, 2019.
- [Wo20] Woltmann, Lucas; Hartmann, Claudio; Habich, Dirk; Lehner, Wolfgang: Best of both worlds: combining traditional and machine learning models for cardinality estimation. In: aiDM@SIGMOD 2020. pp. 4:1–4:8, 2020.
- [Wo21] Woltmann, Lucas; Hartmann, Claudio; Habich, Dirk; Lehner, Wolfgang: Aggregate-based Training Phase for ML-based Cardinality Estimation. In: BTW. pp. 135–154, 2021.
- [Ya20] Yang, Zongheng; Kamsetty, Amog; Luan, Sifei; Liang, Eric; Duan, Yan; Chen, Xi; Stoica, Ion: NeuroCard: One Cardinality Estimator for All Tables. Proc. VLDB Endow., 14(1):61–73, 2020.
- [Zh18] Zhao, Zhuoyue; Christensen, Robert; Li, Feifei; Hu, Xiao; Yi, Ke: Random Sampling over Joins Revisited. In: SIGMOD. pp. 1525–1539, 2018.

# RMG Sort: Radix-Partitioning-Based Multi-GPU Sorting

Ivan Ilic<sup>1</sup>, Ilin Tolovski<sup>2</sup>, Tilmann Rabl<sup>3</sup>

**Abstract:** In recent years, graphics processing units (GPUs) emerged as database accelerators due to their massive parallelism and high-bandwidth memory. Sorting is a core database operation with many applications, such as output ordering, index creation, grouping, and sort-merge joins. Many single-GPU sorting algorithms have been shown to outperform highly parallel CPU algorithms. Today's systems include multiple GPUs with direct high-bandwidth peer-to-peer (P2P) interconnects. However, previous multi-GPU sorting algorithms do not efficiently harness the P2P transfer capability of modern interconnects, such as NVLink and NVSwitch. In this paper, we propose RMG sort, a novel radix partitioning-based multi-GPU sorting algorithm. We present a most-significant-bit partitioning strategy that efficiently utilizes high-speed P2P interconnects while reducing inter-GPU communication. Independent of the number of GPUs, we exchange radix partitions between the GPUs in one all-to-all P2P key swap and achieve nearly-perfect load balancing. We evaluate RMG sort on two modern multi-GPU systems. Our experiments show that RMG sort scales well with the input size and the number of GPUs, outperforming a parallel CPU-based sort by up to 20×. Compared to two state-of-the-art, merge-based, multi-GPU sorting algorithms, we achieve speedups of up to 1.3× and 1.8× across both systems. Excluding the CPU-GPU data transfer times and on eight GPUs, RMG sort outperforms the two merge-based multi-GPU sorting algorithms up to 2.7× and 9.2×.

**Keywords:** Multi-GPU sorting; radix partitioning; high-speed interconnects; database acceleration

## 1 Introduction

Today's data volumes oftentimes exceed the size that database systems can analyze efficiently [Gu15, Ja14]. To improve the data processing performance, research and industry exploit modern hardware. GPUs provide high computational power via thousands of cores, and a high-bandwidth memory [NV17, NV20]. For compute-intensive tasks on small, in-GPU-memory data sets, GPUs achieve orders of magnitude higher instruction throughput (e. g. TFLOPS) than CPUs. Thus, they are commonly used as accelerators for deep learning and HPC workloads [SMY20]. However, GPUs experience a slower adoption into the database systems market, because of the transfer bottleneck [CI18, Lu20]. For many GPU-based operator implementations, copying the data to the GPU and back over the PCIe 3.0 interconnect has been the limiting factor [GK18, Lu20, Ra20, RLT20].

In recent years, high-bandwidth, low-latency interconnects, such as NVIDIA's NVLink, AMD's Infinity Fabric, and the Compute Express Link (CXL) have been introduced [AM18,

<sup>1</sup> Hasso Plattner Institute, University of Potsdam, Germany ivan.ilic@student.hpi.de

<sup>2</sup> Hasso Plattner Institute, University of Potsdam, Germany ilin.tolovski@hpi.de

<sup>3</sup> Hasso Plattner Institute, University of Potsdam, Germany tilmann.rabl@hpi.de

NV18, ST20]. They increase the GPU-interconnect bandwidth close to that of main memory, accelerating CPU-to-GPU and P2P transfers. On hardware platforms with high-speed interconnects, GPUs efficiently accelerate data analytics workloads and core database operations [Lu20, Ma22, Ra20]. Sorting is one such operation, with applications in index creation, duplicate removal, user-specified output ordering, grouping, and sort-merge joins [Gr06]. Over the past years, numerous single-GPU sorting algorithms have been proposed and shown to outperform highly parallel CPU algorithms by orders of magnitude. Parallel radix sort algorithms are best suited for modern GPUs [MG16, SMY20, Ma22].

Modern server-grade systems combine multiple GPUs for an even higher computing power. The research community extended algorithms to utilize multiple GPUs [RLT20, Pa21]. To the best of our knowledge, all published multi-GPU sorting algorithms are sort-merge approaches [GK18, PSHL10, RLT20, Ta13]. The P2P merge sort by Tanasic et al. utilizes inter-GPU communication to merge the previously sorted chunks within GPU memory [Ta13]. The HET merge sort by Gowanlock et al. uses the CPU to merge GPU chunks. Evaluated on modern multi-GPU systems, both algorithms show promising speedups over a single GPU [Ma22]. However, their merging workload increases with the number of GPUs. For HET merge sort, the final multiway merge on the CPU quickly becomes a bottleneck [GK18, Ma22]. For P2P merge sort, scaling up the number of GPUs linearly increases the number of key swaps over the P2P interconnects. During their merge phase, each GPU swaps data with only one other GPU at a time. Thus, multiple merge steps are necessary. This algorithm design made sense in a time when GPUs had no direct P2P interconnects and GPUs communicated with each other via the host-side. On such systems, many concurrent P2P transfers over the PCIe 3.0 tree topology would suffer from shared bandwidth effects and throttle the overall throughput [Ma22]. Today, modern multi-GPU platforms incorporate direct high-bandwidth P2P interconnects. Recent hardware systems support non-blocking all-to-all inter-GPU communication [NV18, NV21b]. In the light of these hardware improvements, we propose RMG sort, a novel radix-partitioning-based multi-GPU sorting algorithm that utilizes the bandwidth of modern P2P interconnects more efficiently. We reduce inter-GPU communication by exchanging the radix partitions between all GPUs once and in parallel, independent of the number of GPUs. Our contributions are:

1. We design a novel multi-GPU sorting algorithm (RMG sort). We employ an MSB radix partitioning strategy to exploit modern P2P interconnects (Section 3).
2. We implement RMG sort in the CUDA framework and publish our source code with automated benchmark scripts to enable reproducible evaluation results<sup>4</sup> (Section 4).
3. We evaluate RMG sort on up to eight GPUs. We compare to parallel CPU-only algorithms and state-of-the-art, merge-based multi-GPU sorting algorithms (Section 5).

---

<sup>4</sup> <https://github.com/hpides/rmg-sort>

## 2 Background

In this section, we explain the required background information about the GPU hardware architecture, modern interconnect technologies, and radix sort algorithms.

**GPU Architecture.** GPUs are designed to support massively parallel computations, hiding memory access latency with concurrently executed computation [NV22b]. They are equipped with thousands of cores that are organized in a specialized hierarchy. The main unit of computation is the streaming multiprocessor (SM) [NV20], equivalent to the compute unit for AMD GPUs [AM20]. One GPU consists of an array of SMs [NV17, NV20]. Each SM can run multiple concurrent groups of threads (thread blocks). A thread block can run up to 1024 threads. The SM schedules these high numbers of threads in groups of 32 consecutive threads, so-called warps, that execute the same instruction. GPUs excel at achieving high instruction throughput rates and provide a high-bandwidth memory. The memory hierarchy is divided into *off-chip* and *on-chip* memory. Off-chip memory mainly consists of global HBM2 memory which all running threads access. It provides peak bandwidth rates of up to 1555 GB/s [NV20]. Compared to main memory, the GPU memory capacity is limited (up to 80 GB). The GPU's L2 cache hides the latency of global memory accesses. In addition, each SM comes with a local, high-bandwidth, low-latency L1 cache to accelerate computation on frequently used data. While the L1 cache automatically hides accesses of all threads of its SM, shared memory needs to be explicitly managed by the programmer.

**GPU Interconnects.** GPUs are attached to the CPU memory controller via an interconnect. The interconnect topology significantly impacts the performance of multi-GPU applications [Li20]. In the following, we explain modern interconnect technologies. PCIe 3.0 is used as the standard interconnect for many peripheral devices, including GPUs. It supports full-duplex communication at 16 GB/s per direction. PCIe 4.0 doubles this bandwidth rate for a theoretical peak of 32 GB/s. Multi-GPU systems with no direct P2P interconnects only support P2P communication through multi-hop host-side transfers. Over the last few years, hardware vendors introduced high-bandwidth, low-latency GPU interconnects for direct P2P transfers. AMD released the Infinity Fabric interconnect [AM18], while NVIDIA launched NVLink. NVLink 2.0 achieves 25 GB/s per link per direction. One NVLink 2.0-enabled GPU supports six links for a theoretical peak bandwidth of 150 GB/s per direction. NVLink 3.0 doubles the number of links per GPU for a bandwidth of 300 GB/s. NVLink is primarily designed to accelerate inter-GPU communication. NVSwitch is an NVLink-based switch chip by NVIDIA that enables non-blocking, all-to-all, inter-GPU communication at high bandwidth. It connects up to 16 GPUs between each other in a point-to-point mesh [NV18].

**Radix Sort.** Radix sort is a non-comparison-based sorting algorithm with linear computational complexity [Ag96, Gi19, SJ17]. Radix sort algorithms iterate over the keys' bits and partition the keys into distinct buckets based on their radix value. To reduce the number of iterations, radix sort algorithms look at multiple consecutive bits  $c$  at a time. Typically, a radix sort algorithm either starts from the most or the least significant bit (MSB or LSB). Given  $k$ -bit keys, the number of partitioning passes is  $p = \lceil k/c \rceil$ . In each partitioning

pass, each of the  $n$  input keys is scattered into one of  $2^c$  distinct buckets depending on the currently considered  $c$  bits until all  $k$  bits have been considered, i. e. the keys are sorted. This leaves radix sort with a computational complexity of  $O(n \times p)$ . An LSB radix sort algorithm stores the  $2^c$  buckets of the current partitioning pass only, as long as it respects the keys' sort order from preceding rounds. In contrast, an MSB radix sort algorithm refines the partitioning within each bucket in each round, keeping track of increasing numbers of buckets. To scatter the keys into their corresponding buckets in parallel, many GPU-based radix sort algorithms operate out-of-place [Sa10, SJ17, ZB91, ZW12].

### 3 Algorithm

In this section, we explain our radix-partitioning-based multi-GPU sorting algorithm (RMG sort). It sorts the input keys using only the GPUs. Therefore, it only sorts data sets that fit into the combined device memory of the system's GPUs. We use a most significant bit (MSB) radix partitioning strategy. Our algorithm requires one all-to-all key swap between the GPUs over the P2P interconnects, independent of the number of GPUs. Our algorithm reduces the inter-GPU communication compared to previous sort-merge algorithms. In summary, RMG sort works as follows: First, the unsorted input keys are copied to the GPUs in chunks of equal size. Each GPU partitions its keys locally, starting from the most significant bit, until every radix bucket on each GPU is *small enough* for the following all-to-all P2P key swap between the GPUs. The P2P key swap re-distributes all buckets across all GPUs so that afterwards, 1) each GPU contains keys of a distinct value range and 2) bringing all keys into the global sort order across the  $g$  GPUs does not require any further key swaps. In other words, after the P2P key swap, all keys of GPU  $i$  have smaller or equal most significant bits compared to the keys of GPU  $i + 1$ . Then, each GPU sorts its buckets locally to bring the keys across all  $g$  GPUs into the final sorted order.

This allows for two optimizations: First, we reduce the final sorting workload. Instead of sorting the entire chunk, each GPU sorts its radix buckets individually. Given that the partitioning phase already examined the most significant  $r$  bits of each key, we sort each bucket on the remaining  $k - r$  bits. Secondly, we interleave the sorting computation with copying the data back to the CPU. Once a bucket is fully sorted, we transfer it back, while the remaining buckets are still being sorted. Thereby, we hide the time duration of the sorting computation on the GPUs. In the following sections, we explain how the radix partitioning phase ensures that one bucket exchange (P2P key swap) between the GPUs is sufficient, even for skewed data. We outline how we distribute the keys across the GPUs with nearly perfect load balancing and how we accelerate the final sorting computation.

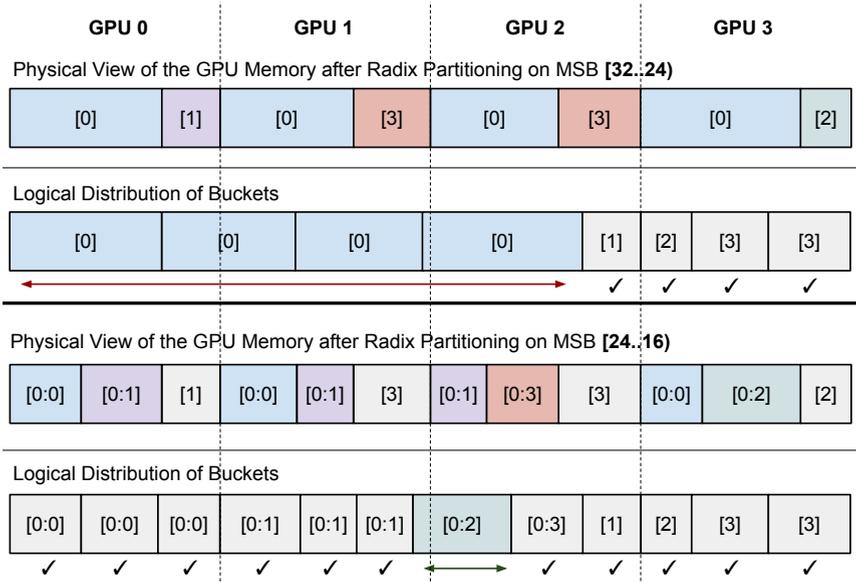
#### 3.1 On-GPU MSB Radix Partitioning

After the  $n$  input keys are copied to the  $g$  GPUs in equal sized chunks, each GPU partition its keys locally (i. e. in its own device memory). Each GPU first computes the histogram over

its  $\lceil n/g \rceil$  keys on the most significant  $c$  bits. Calculating the prefix sum on the histogram returns the starting write offsets for each of the  $2^c$  buckets. Using the prefix sum, each GPU partitions its chunk in device memory so that all keys of bucket  $i$  precede bucket  $i + 1$ .

For most data distributions, the probability is high that there is a radix bucket for which every GPU finds associated keys. In fact, for uniformly distributed keys, every GPU likely contains keys that belong to every one of the  $2^c$  possible buckets. The goal of the P2P key swap is to re-distribute the keys across all  $g$  GPUs so that all keys that belong to the same bucket are aligned in the device memory of one and the same GPU. We also have to ensure that all keys of GPU  $i$  are smaller than or equal to the ones on GPU  $i + 1$ . We satisfy both constraints by distributing the keys across the GPUs in the order of their radix digit values, i. e. by distributing the buckets in ascending order: The buckets of the smallest radix values to GPU 0, and those with the highest radix values to GPU  $g-1$ . After each partitioning pass, each GPU sends its histogram to all other GPUs via the P2P interconnects. Thus, each GPU knows about the entire key distribution and computes the logical distribution of buckets, i. e. the placement of buckets across the  $g$  GPUs in ascending order. Here, we check whether the current level of partitioning allows for each complete bucket to fit onto its designated GPU. A bucket  $b$  on GPU  $i$  is *complete* if all of the keys that reside on GPU  $i$  and that fall into bucket  $b$  are aligned at subsequent addresses in the memory of GPU  $i$ . Given  $g$  GPUs, each GPU might produce a complete bucket  $b$ . A set of at most  $g$  complete buckets  $[b]_0, [b]_1, \dots, [b]_{g-1}$  forms a spanning bucket under a given logical bucket distribution if all keys that belong to bucket  $b$  will not fit into the memory of the designated GPU. A spanning bucket prevents us from performing the P2P bucket exchange because we could not fully sort the spanning bucket without further communication between those GPUs that the bucket spans. We need to refine each spanning bucket in subsequent partitioning passes.

The number of partitioning passes necessary depends on the distribution of input keys. The input data might be highly skewed and contain only leading zeros in the most significant  $c$  bits. It is desirable for our partitioning phase to split the keys into reasonably small buckets to avoid load imbalances between the GPUs. We perform multiple partitioning passes on subsequent sets of  $c$  bits, starting from the most significant one, until there are no spanning buckets left. Any subsequent partitioning pass refines only the spanning buckets while the buckets that already fit onto one GPU stay untouched. In Figure 1, we show an example of our radix partitioning algorithm on four GPUs. In the example, we sort 32-bit keys while considering  $c = 8$  bits at a time. In the first pass, each GPU scatters its keys based on the bits [32..24). We show the result of the local partitioning step in the top half of each pass, i. e. the physical view of the GPU memory. All GPUs find many keys that belong to bucket 0. They exchange their histogram information which allows each GPU to construct the logical distribution of complete buckets, shown in the bottom half of each pass in Figure 1. The complete bucket [0] is a spanning bucket after the first partitioning pass. Consequently, we continue with another pass on bits [24..16) on the spanning bucket [0]. In the second partitioning pass, each GPU with keys of the spanning bucket [0] refines its part (e. g. into two smaller buckets [0:0] and [0:1] on GPU 0). After the histogram exchange of the second



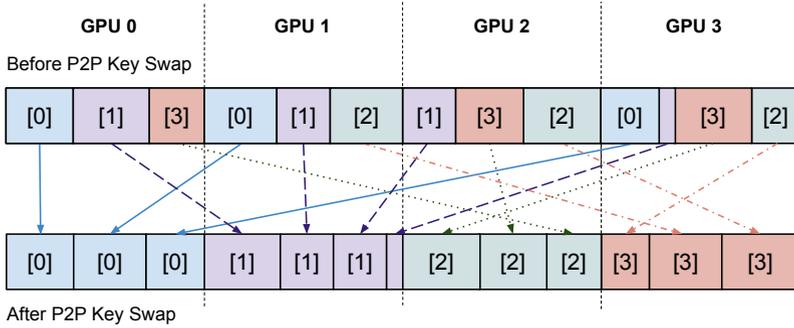
**Fig. 1: Radix partitioning phase example**

pass, we compute the logical bucket distribution and find that we resolved the spanning bucket. Bucket [0:2] (resident on GPU 3) is not a spanning bucket because we allow for nearly-perfect load balancing. Thus, our radix partitioning phase is completed.

### 3.2 Multi-GPU P2P Bucket Exchange

The bottom of Figure 1 shows an example of a final, logical bucket distribution. It aligns the complete buckets in globally sorted order across the GPUs. After the partitioning phase, the keys of each bucket still reside in the memory of their initial GPU as they have only been partitioned locally. In the multi-GPU P2P key swap, we re-distribute them between all GPUs. Each bucket’s destination GPU can either be the same as the source GPU or a remote GPU in which case the memory copy goes over the P2P interconnects. For the P2P bucket exchange, the logical bucket distribution computed from the histogram broadcast of the final partitioning pass is sufficient. We measure that the overhead of calling one asynchronous copy per bucket is negligible even for high numbers of buckets. The CUDA driver appends calls into the CUDA stream queue and performs the copies at peak bandwidth.

**Load Balancing.** To reduce the number of partitioning passes, we do not enforce perfect load balancing. Instead, certain GPUs can handle slightly more keys than others. The first partitioning pass very rarely results in a bucket distribution that is perfectly aligned with



**Fig. 2: P2P bucket exchange example**

the chunk size. We define a threshold  $\epsilon$  as the number of keys that each GPU can use as additional padding at the start and the end of its chunk buffer. This avoids treating slightly overflowing buckets as spanning buckets. Whenever a bucket overflows into a GPU by a number of keys  $\sigma \leq \epsilon$ , these  $\sigma$  keys are assigned to the adjacent GPU that already holds keys of that same bucket. If an overlapping bucket would span over two or three GPUs for a perfectly load-balanced approach, our additional  $\epsilon$ -padding can, in the best case, avoid an entire partitioning pass. If the spanning bucket spans over more than three GPUs, our nearly-perfect load balancing approach reduces the number of GPUs that the bucket spans by up to two. This is because each GPU employs the  $\epsilon$ -padding at the start and the end of its chunk. We empirically determine an optimal  $\epsilon$ -padding of 0.5% of the initial GPU chunk size. With this  $\epsilon$ , we measure that uniformly distributed keys require one partitioning pass.

For extremely skewed distributions, spanning buckets can occur after the last partitioning pass on the least significant  $c$  bits (e. g. if all  $n$  keys are of the same value). Having considered all  $k$  bits after the radix partitioning phase, there will still be one spanning bucket with  $n$  keys over all  $g$  GPUs. In fact, any spanning bucket that remains after the last partitioning pass must consist of keys of the same single value. Thus, we can choose arbitrary borders for where to split the spanning bucket. We simply distribute the keys of each last-pass spanning bucket in a perfectly load-balanced manner across the involved GPUs. Since we employ MSB radix partitioning, the number of buckets grows continuously with each partitioning pass. Considering  $c$  bits at a time, partitioning one spanning bucket divides it into  $2^c$  sub-buckets. We view the initial input of  $n$  keys on the  $g$  GPUs as the initial spanning bucket. A GPU can be involved in at most two spanning buckets (one per adjacent GPU). Thus, the maximum possible number of spanning buckets per partitioning pass is  $g - 1$ . In that case, each spanning bucket spans over two GPUs. In total, we perform a maximum of  $p$  partitioning passes, with  $p = \lceil k/c \rceil$ . In the first pass, we partition the input data as one spanning bucket. This leaves us with an upper bound for the spanning buckets of  $s_{max} = (g - 1) \cdot (p - 1) + 1$ . The total number of buckets can not exceed  $2^c \cdot s_{max}$ . For  $c = 8$ , 64-bit keys and eight GPUs, this is equal to  $256 \cdot s_{max} = 12.545$ .

### 3.3 On-GPU Bucket Sorting

After the P2P key swap, each GPU contains buckets of distinct value ranges. While the different buckets are correctly ordered by their MSB values, the keys within each bucket are still unsorted. To sort each bucket, we use the single-GPU LSB radix sort algorithm `cuda::DeviceRadixSort::SortKeys`, provided in NVIDIA’s CUB library as it achieves the fastest performance [NV21a, Ma22]. Depending on the number of partitioning passes performed, we have already examined a certain number of most significant bits of each key. We use this information to accelerate the sorting computation. For each bucket, we specify the bit range that the local radix sort sorts on. The final bucket partitioning level is heterogeneous in the following sense: Some buckets are sufficiently partitioned after the first pass while others are refined through multiple passes. As a consequence, for each bucket that we sort, we have taken a different number of most significant bits into account during the partitioning phase. Since we store the partitioning pass  $p_b$  that generated each bucket, we determine the bit range to sort on as follows:  $[endbit..0]$ , with  $endbit = k - ((p_b + 1) \cdot c)$ . If a bucket went through the maximum number of partitioning passes  $p$ , we do not need to sort the bucket at all. Compared to sorting on all bits, specifying a reduced bit range improves the sorting performance of the local radix sort significantly. We measure a speedups between 30% and 200% for one, two and three partitioning passes on the NVIDIA Tesla V100 GPU.

The overhead of a single kernel launch is insignificant. When calling one kernel per bucket to sort, the launch times add up. In order to reduce the total number of buckets and mitigate the associated kernel launch overhead, we fuse neighbouring buckets whose number of keys is below a certain threshold. We can only fuse neighbouring buckets because we have to preserve the buckets’ global sort order. We configure the threshold equal to 1% of the initial GPU chunk size. Whenever we fuse two buckets, the bit range that we sort the combined bucket on needs to be extended. To avoid extending the bit range too much, and thereby losing the benefit of the reduced sorting duration, we only fuse buckets of the same partitioning pass. As a result, the combined buckets share their initial bit range  $[endbit..0]$  which we extend by the necessary minimum, i. e. by the most significant bit position in which the two bucket values differ. After the buckets have been fused and each final bucket’s bit range is determined, each GPU sorts its buckets individually. As soon as a bucket is sorted, we transfer it back to the CPU, asynchronously launching the memory copy that transfers the latest sorted range of keys. This approach effectively overlaps the sorting computation with the device-to-host copy operation.

## 4 Implementation

In this section, we explain how we implement our MSB radix partitioning phase. Each partitioning pass includes computing histograms and scattering keys accordingly.

## 4.1 Histogram Computation

After the input data is copied to the GPUs, each GPU computes the histogram on the most significant  $c$  bits of its keys. To compute the histogram, we read all keys of the chunk, and atomically increment one of the  $2^c$  bucket counters in our histogram array depending on the key's radix. When parallelizing the computation, we assign each thread block an equal number of keys to process. To achieve peak performance, each thread block first computes a block-local histogram stored in its shared memory. Atomic operations on shared memory are significantly faster than on global memory. After writing each block-local histogram back to global memory, we need to aggregate these partial histograms into the final GPU-global histogram. For this, we implement a second kernel function that reads the block-local histograms from memory and performs global atomic operations. To reduce the number of global atomics, we launch enough threads for each to pre-aggregate a fixed-sized group of block-local histograms. The resulting read pattern to the block-local histograms is perfectly warp-aligned because we orchestrate the memory accesses based on the thread-id.

For very skewed distributions, the shared memory atomics on the block-local histogram are under increased pressure. In the most extreme case, all keys processed by a thread block have the same  $c$  bits. Thus, all threads try to increase the same bucket counter of the block-local histogram concurrently. To mitigate this performance degradation, we employ the following lightweight optimization: Each thread stores its first key's bucket value and holds back the atomic increment. For every following key, we check if it falls in the same bucket as the first key, and if so, we increment a local variable. After the thread iterated over its keys, we perform the postponed shared atomic increment. With this optimization, our histogram computation is equally fast on skewed and uniform data.

## 4.2 Key Scattering

In the key scattering step, each GPU locally partitions its keys based on the computed histogram, i. e. based on the considered  $c$  bits. Afterward, all keys of bucket  $i$  precede those of bucket  $i + 1$ . To avoid synchronization between reading from and writing to the same memory buffer with many threads, we perform the key scattering step out-of-place. We allocate two alternating input/output buffers. Depending on the bucket, each key needs to be scattered to different locations in global memory. To avoid random write patterns, we pre-scatter all the keys of a thread block into their respective buckets in shared memory. This allows each thread block to write its buckets back to global memory one after another. As a result, the write pattern is sequential for keys of the same bucket. We launch the same number of threads and thread blocks for our `ScatterKeys` kernel as for the histogram computation. Thus, we can re-use the block-local histograms. For both, the pre-scattering and the global write-back, we know the write position for each key of a bucket. In both cases, we determine the write positions by computing the prefix sum on the corresponding histogram, i. e. either the GPU-global or a block-local one.

**Shared Memory Pre-Scattering.** Each thread block, scheduled to one SM, is responsible for pre-scattering its share of keys into its shared memory buffer. First, the thread block loads its block-local histogram and computes the prefix sum on it. This gives us the starting write position  $w_{sh}$  for each bucket  $b$ , i. e. the write position for the first key of  $b$ . Then, each thread iterates over its assigned keys, determining their buckets. For a key of bucket  $b$ , we perform an `atomicAdd` operation that reads  $w_{sh}[b]$  and adds 1. We then scatter the key into the shared memory buffer at the position that we read from  $w_{sh}[b]$ . Thus, we ensure that any subsequent writes by another or the same thread will be performed on the incremented offset, avoiding write conflicts. We deliberately perform significantly more atomic operations in shared memory than in global memory because they are substantially faster in shared memory. The limited shared memory size (128 KB on the Tesla V100) sets an upper bound for the number of keys that each thread block can pre-scatter. We configure our implementation to process twelve 32-bit keys per thread, and each thread block to run 1024 threads. For 64-bit keys, each thread processes six keys. This results in a shared memory usage of 49.152 KB, leaving enough memory to be used as the L1 cache.

**Global Memory Write-Back.** We compute the prefix sum on the GPU-global histogram before calling the `ScatterKeys` kernel. It returns the starting write position to the global memory output buffer  $w_{gl}$  for each complete bucket. Since all thread blocks concurrently write their share of keys back to global memory, we pre-determine the exact global memory buffer spaces that each thread block writes to. For this, every thread block atomically increments  $w_{gl}[b]$  by the number of keys it will write per bucket  $b$ . This ensures that the global memory write-back phase needs no further synchronization. The pre-scattering step enables sequential writes. But to achieve peak memory throughput, the write pattern needs to be warp-aligned. We implement each thread warp to be responsible for writing a small, constant number of consecutive buckets, one after the other. Each thread of a warp writes one of 32 consecutive keys of the warp's current bucket, iterating over the bucket's keys with a stride of 32 (= thread warp size [NV22a]). With this approach, the bucket size can negatively influence the memory throughput. If many buckets contained very few keys, the memory throughput would drop considerably as many threads idle. It is desirable that all non-empty buckets contain enough keys to fill at least one thread warp memory transaction. We cannot increase the number of keys per bucket by processing more keys per thread block because the shared memory size is limited. Rather, the number of consecutive bits  $c$  considered per partitioning pass influences how many keys can fall into a bucket. For example, if we chose  $c=16$ , the number of possible buckets  $2^{16}$  would be higher than the number of keys one thread block processes. This would drastically reduce the memory throughput since most of the buckets would contain very few or no keys, assuming uniform distributions. If  $c$  is too small, we increase the number of partitioning passes, ultimately increasing the total sort duration. We configure  $c=8$  as an ideal trade-off between minimizing the number of partitioning passes and maximizing the throughput. Thus, we confirm the findings of Stehle et al. for their single-GPU MSB radix sort [SJ17]. We measure a global write-back throughput of 70-95% of the A100 GPU's peak memory bandwidth.

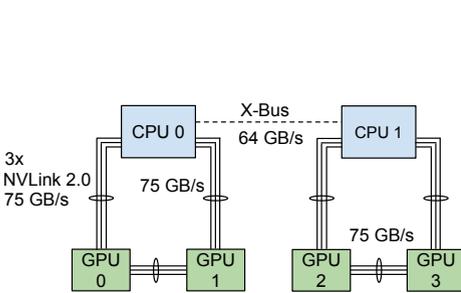
## 5 Evaluation

In this section, we compare the performance of RMG sort to highly parallel CPU algorithms (Section 5.1), and two state-of-the-art merge-based multi-GPU sorting algorithms (Section 5.2). We analyze RMG sort for varying distributions and data types in Section 5.3.

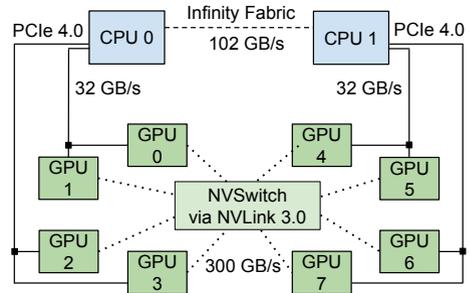
**Experimental Setup.** We evaluate our algorithm on two multi-GPU platforms with state-of-the-art interconnects. We provide system information in Table 1. The IBM Power AC922 is a two-socket system that attaches two NVIDIA Tesla V100 GPUs to each NUMA node [NSH18] (Figure 3, bandwidth per direction). On the IBM AC922, NVLink 2.0 accelerates data transfers between a NUMA node and its two GPUs. Our second system is the DGX A100 [NV21b]. It connects eight NVIDIA A100 GPUs with NVLink 3.0-based NVSwitch for high-speed transfers between all GPUs at 300 GB/s per direction (Figure 4).

**Tab. 1: Hardware systems overview**

|       | (a) IBM Power System AC922       | (b) NVIDIA DGX A100                 |
|-------|----------------------------------|-------------------------------------|
| CPU   | 2× IBM POWER9 à 16 cores 2.7 GHz | 2× AMD EPYC 7742 à 64 cores 2.3 GHz |
| GPUs  | 4× NVIDIA Tesla V100 SXM2 32 GB  | 8× NVIDIA A100 SXM4 40 GB           |
| RAM   | 2× 256 GB DDR4                   | 2× 512 GB DDR4                      |
| Tools | CUDA 11.2, GCC 10.2.1            | CUDA 11.4, GCC 9.3.0                |



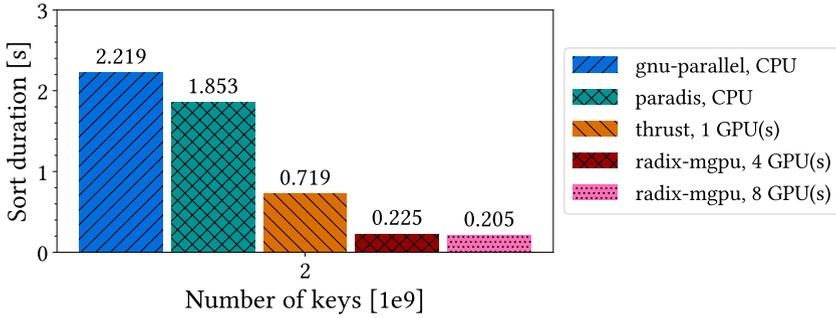
**Fig. 3: IBM AC922 topology**



**Fig. 4: NVIDIA DGX A100 topology**

For all experiments, we measure the end-to-end sort duration which includes the data transfer times between CPU and GPUs. We run every experiment five times and report the arithmetic mean. Our experiments results are stable with a standard error across all runs of less than 4% from the mean. We assume that the input data is not distributed perfectly among the NUMA nodes. Instead, it lies in the main memory of NUMA node 0 only. An optimized NUMA strategy would benefit all three evaluated multi-GPU sorting algorithms equally, and is to consider in future work. We pre-allocate the GPU memory and the pinned host memory because we assume exclusively reserved accelerators. We publish the source code of RMG sort with benchmark scripts to automatically run and plot the experiments.

**Optimal GPU Sets.** Given a fixed number of GPUs  $g$  with  $g \in \{1, \dots, g_{max}\}$ , the interconnect topology determines which exact  $g$  GPUs achieve the fastest sorting execution. For instance, when using a P2P-based algorithm on the IBM AC922, the optimal two-GPU-set is the



**Fig. 5: Sorting performance: CPU vs. GPU(s) on the DGX A100**

CPU-local GPU pair (0, 1) (for NUMA node 0). The optimal four-GPU-set on the DGX A100 is (0, 2, 4, 6) as it includes only one GPU of each pair that shares a PCIe switch. Across our evaluation, we depict the performance for optimal GPU sets.

**Baselines.** To compare the performance of RMG sort to that of the CPU, we use the state-of-the-art parallel CPU radix sort PARADIS as our baseline [Ch15]. We add the parallel multiway merge sort from the GNU parallel algorithms as our second CPU baseline [FS21]. To compare the performance of multiple GPUs against one, we use the same primitive as in RMG sort, namely `cub::DeviceRadixSort` [NV21a]. This LSB radix sort is, to the best of our knowledge, the fastest single-GPU sorting algorithm [Ma22]. As such, it is integrated into NVIDIA’s Thrust library as the standard sorting method `thrust::sort` [NV21d].

**Memory consumption.** Since CUB’s device radix sort works out-of-place, RMG sort and the two multi-GPU sorts we compare to have a memory consumption of at least  $2n$  for  $n$  input keys. For RMG sort, we additionally store histograms and bucket mapping tables. The additional memory overhead of RMG sort depends on the upper bound of spanning buckets, and thus increases with the number of GPUs and partitioning passes. In our implementation, sorting 16 billion 32-bit keys with eight GPUs requires an additional memory overhead of 3.7 GB – 22% of the  $2n$  memory overhead (=16 GB). When sorting 8B keys on four GPUs, the additional overhead is 11%. However, as part of future work, we want to improve our implementation and reduce the additional overhead significantly by re-using buffers in each partitioning pass. Then, the additional memory overhead only depends on the number of GPUs. In the above cases, it would constitute 7%, and 3.5%, respectively.

## 5.1 CPU Comparison

First, we compare the performance of RMG sort to the CPU. We sort two billion uniformly distributed 32-bit integer keys with our proposed RMG sort, and our single-GPU and CPU baselines. We depict the results for the NVIDIA DGX A100 in Figure 5. The system’s optimal four-GPU-set is (0, 2, 4, 6). We observe that one GPU achieves a speedup of  $3\times$

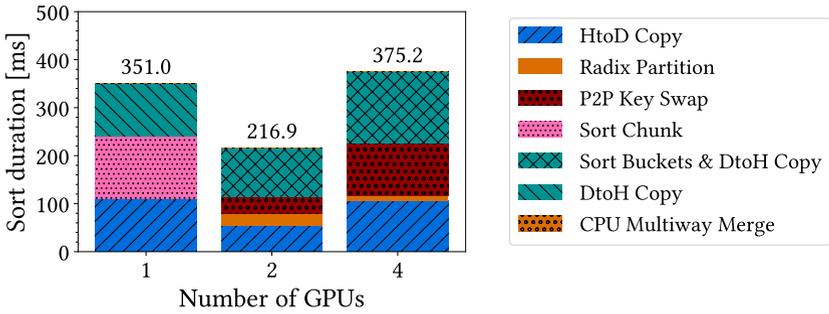
over `gnu-parallel` and  $2.6\times$  over PARADIS. Given that radix sort algorithms are memory-bandwidth-bound, one main reason why the GPU outperforms the CPU is the GPU’s substantially higher memory bandwidth. Compared to `gnu-parallel`, RMG sort is  $9.8\times$  faster with four GPUs and  $10.8\times$  with eight. Also, RMG sort outperforms PARADIS  $8\times$  with four GPUs, and  $9\times$  with eight. Overall, eight GPUs achieve the best performance. On the IBM AC922, we measure that RMG sort outperforms the CPU up to  $20\times$ . The speedup is higher than on the DGX A100 because the POWER9 CPU has  $4\times$  fewer cores than the AMD EPYC 7742 and the IBM AC922 achieves faster CPU-GPU copies with NVLink 2.0.

## 5.2 Radix-Partitioning vs. Sort-Merge

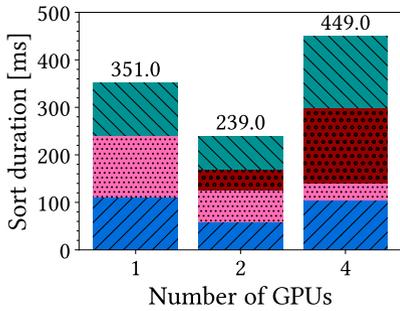
In this section, we compare the performance of RMG sort to two state-of-the-art merge-based multi-GPU sorting algorithms, the **P2P merge sort** by Tanasic et al. [Ta13], and the heterogeneous merge sort (**HET merge sort**) by Gowanlock et al. [GK18]. P2P merge sort sorts and merges on multiple GPUs using P2P interconnects. By selecting a pivot within the sorted chunks of a GPU pair, blocks of keys are swapped so that the first GPU contains keys smaller than or equal to the keys of the second GPU. Merging the two blocks of keys on each GPU locally brings the data across both GPUs into globally sorted order. The algorithm sorts on more than two GPUs using many subsequent P2P key swaps and GPU-local merge steps. The number of P2P transfers scales linearly with the number of GPUs  $g$ . P2P merge sort can only sort on  $g = 2^k$  GPUs,  $k \in \mathbb{N}$ . RMG sort runs on any number of GPUs. HET merge sort uses a parallel multiway merge algorithm on the CPU to merge chunks that the GPUs have sorted, and is not limited by the combined GPU memory capacity. Since RMG sort only sorts data that fits onto the GPUs, we disregard the evaluation of out-of-core data.

To evaluate the performance differences between RMG sort and the merge-based algorithms, we break down the total sort duration of each algorithm into phases. All three algorithms start with the host-to-device (HtoD) copy. For RMG sort, the remaining phases are the radix partitioning, the P2P key swap, and the bucket sorting phase which is interleaved with copying the buckets back to the host. For P2P merge sort, we analyze the HtoD copy, the sort phase, the P2P merge phase on the GPUs, and the device-to-host copy (DtoH). In its sort phase, each GPU chunk is sorted using CUB’s single-GPU radix sort. HET merge sort entails the same phases as P2P merge sort except for its CPU-based multiway merge.

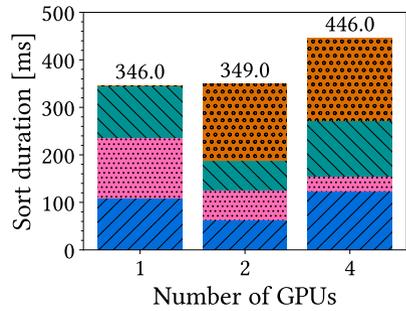
**Sort Duration Breakdown – IBM AC922.** In Figure 6, we depict the sort duration breakdown for RMG sort, P2P merge sort, and HET merge sort for 2B (two billion) uniformly distributed 32-bit integer keys on the IBM AC922. We depict the results for the single-GPU baseline, the GPU pair (0, 1), and all four GPUs. Both RMG sort and P2P merge sort exchange keys via the P2P interconnects. Since the merge phase of P2P merge sort is bound by the P2P key swaps and not the GPU-local merge steps, we display the P2P merge phase using the same plot label and bar pattern as for RMG sort’s P2P key swap. We observe that the radix partitioning achieves the shortest duration out of all algorithm phases, scaling linearly to the number of keys, i. e. the GPU chunk size. On two GPUs it makes



(a) RMG sort



(b) P2P merge sort



(c) HET merge sort

**Fig. 6: Sort duration breakdown: Sorting two billion integer keys on the IBM AC922**

up 11% of RMG sort’s total sort time with 22.8ms, while it takes four GPUs 11.4ms. For the GPU pair (0, 1), the second shortest time duration is the P2P key swap. Powered by a bandwidth of 75 GB/s, the P2P bucket exchange takes 36ms (16% of the total sort duration). While the HtoD copy is halved compared to the single-GPU baseline, the *Sort Buckets & DtoH Copy* phase makes up 47% of the total sort duration. This is because the DtoH copy throughput drops by 30% compared to HtoD copies for parallel transfers from multiple GPUs to the same NUMA node [Ma22]. In addition to this hardware anomaly, we measure that the sort-copy-overlap does not perfectly hide the sorting time. Still, overlapping the sorting computation with the DtoH copy saves 50% of the sorting time (=32ms) on two GPUs. This explains why RMG sort achieves a speedup of 1.6× with two GPUs over one.

Figure 6a also shows why four GPUs perform worse than two on the IBM AC922. The CPU-interconnect is the system’s transfer bottleneck, as the X-Bus reaches only 41 GB/s of the theoretical peak bandwidth of 64 GB/s [Pe19, Ma22]. Thus, the slow CPU-GPU copies on the remote GPUs 2 and 3 slow down the execution. Also, the P2P throughput suffers from the low X-Bus bandwidth, as the P2P key swap takes 3× longer on four GPUs than on two. As a result, RMG sort performs 7% slower on four GPUs compared to one GPU. In Figure 6b, we observe that the number of P2P key swaps of P2P merge sort scales linearly

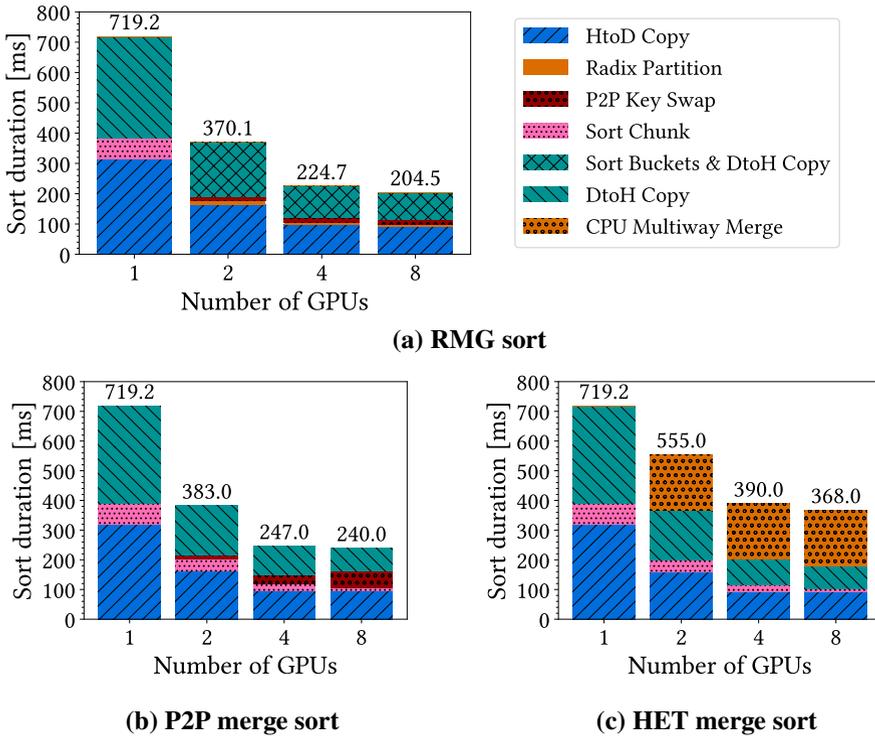
with the number of GPUs. On two GPUs, P2P merge sort requires only a single key swap, similar to RMG sort, which results in nearly identical transfer times. However, since we overlap the sorting computation with the DtoH copy, RMG sort outperforms P2P merge sort on two GPUs by 11%. When comparing RMG sort with HET merge sort (see Figure 6c), we observe that the CPU multiway merge is significantly slower compared to our on-GPU partitioning. On two GPUs, it takes RMG sort 2.7× less time to partition and swap the keys than it takes the CPU to merge the two sorted chunks. On four GPUs, the P2P throughput of RMG sort is significantly reduced. Still, the radix partitioning and the P2P key swap phase combine for a time duration that is 44% lower than that of the CPU merge. In total, RMG sort outperforms HET merge sort 1.6× on two, and 1.2× on four GPUs for 2B keys.

We conclude that on GPUs with high-bandwidth P2P interconnects, GPU-only approaches like RMG sort and P2P merge sort are superior to the CPU-based merge. RMG sort reduces the inter-GPU communication compared to P2P merge sort only for  $g > 2$ . On this system, where two NUMA-local GPUs are optimal, we outperform P2P merge sort because we overlap the sorting computation with the DtoH copy – an optimization of our MSB radix partitioning. P2P merge sort waits for the last key to be merged before the DtoH copy.

**Sorting Large Data – IBM AC922.** We observe that the speedup of RMG sort over P2P merge sort increases with larger inputs. RMG outperforms P2P merge sort by 11% for 2B keys, and by 17% for 4B keys. We outperform HET merge sort 1.6× for 2B keys, and 1.7× for 4B keys. Sorting 12B integer keys (48 GB) with four GPUs on the IBM AC922, RMG sort outperforms HET merge sort by 2.1×. P2P merge sort cannot sort more than  $2^{32}$  keys per GPU due to an input size limitation in its on-GPU merge implementation.

**Sort Duration Breakdown – DGX A100.** Figure 7 depicts the sort duration breakdown sorting 2B (two billion) uniformly distributed keys on the DGX A100. We evaluate the GPU sets (0, 2), (0, 2, 4, 6), and all eight. First, we note that the HtoD and DtoH copies take significantly longer on this system than on the IBM AC922 due to the comparatively low PCIe 4.0 bandwidth. For RMG sort, the CPU-GPU transfers make up 90% of the end-to-end duration. In Figure 7a, we observe that our partitioning phase scales well to increasing numbers of GPUs. It takes 14.4ms on two GPUs, 7.2ms on four GPUs, and 3.6ms on eight GPUs, which constitutes 4%, 3%, and 2% of the total sort duration, respectively. The P2P key swap time stays constant, independent of the number of GPUs. Given the high bandwidth of NVLink 3.0. We measure 14-17ms for two, four and eight GPUs (less than 8% of the total sorting time). The P2P swap time is not reduced when increasing the number of GPUs even though each GPU copies less data. Even though NVSwitch does achieve simultaneous all-to-all transfers at high throughput, P2P transfers between individual pairs of GPUs tend to perform best. Similar to our partitioning phase, the time of the sorting computation gets halved when we double the number of GPUs. However, the sorting time on the A100 GPU is insignificant compared to the HtoD/DtoH copies. Thus, the sort-copy-overlap does not notably improve the end-to-end performance on this system.

Despite the CPU-GPU copy bottleneck on the DGX A100, we observe RMG sort to scale comparatively well from one to eight GPUs. Compared to a single GPU, we achieve



**Fig. 7: Sort duration breakdown: Sorting two billion integer keys on the DGX A100**

speedups of  $1.9\times$  with two,  $3.2\times$  with four, and  $3.5\times$  with eight GPUs. We cannot expect much higher speedups on eight GPUs because of the shared bandwidth effects that result from the system's limited number of PCIe switches. As seen in Figure 4, neighbouring pairs of GPUs share a PCIe switch. For parallel CPU-GPU transfers, the throughput for neighbouring GPU pairs cannot exceed the 32 GB/s of one PCIe 4.0 instance. This hardware limitation negatively influences any multi-GPU sorting algorithm, not just RMG sort. In Figure 7b, we again see that the P2P merge phase time of P2P merge sort increases with the number of GPUs. We measure it to take almost  $4\times$  longer when eight GPUs merge their chunks compared to two. This explains why RMG sort's speedup factor over P2P merge sort increases:  $3\%$  with  $g=2$ ,  $10\%$  with  $g=4$ , and  $17\%$  with  $g=8$ . When RMG sort uses eight GPUs, the radix partitioning phase and the P2P key swap take 20ms, which is  $2.7\times$  less than the P2P merge phase takes. In Figure 7c, we again observe the CPU merge as the limiting factor of HET merge sort. Compared to the combined duration of RMG sort's partitioning phase and its P2P swap on two, four, and eight GPUs, the CPU merge takes  $6.6\text{--}9.2\times$  longer. In total, RMG sort outperforms HET merge sort up to  $1.8\times$  on eight GPUs. Thus, if we compare the execution times for the on-GPU (or on-CPU) computation and the P2P transfers only, i. e. excluding the HtoD and DtoH copies, RMG sort outperforms

**Tab. 2: Sorting performance by data distribution (2 billion keys, 8 GPUs, DGX A100)**

|                       | zero  | sorted | nearly-sorted | reverse-sorted | uniform | normal |
|-----------------------|-------|--------|---------------|----------------|---------|--------|
| <b>RMG sort</b>       | 182ms | 190ms  | 192ms         | 193ms          | 205ms   | 215ms  |
| <b>P2P merge sort</b> | 182ms | 189ms  | 200ms         | 250ms          | 240ms   | 243ms  |

P2P merge sort 2.7 $\times$ , and HET merge sort 9.2 $\times$  with eight GPUs. The slow HtoD and DtoH copies reduce the total end-to-end speedup. Still, we demonstrate the potential speedup that RMG sort could achieve if the system included high-speed CPU-GPU interconnects. For the DGX A100, we confirm that P2P-based multi-GPU approaches sort significantly faster than the heterogeneous strategies. We conclude that, compared to P2P merge sort, RMG sort more efficiently utilizes the non-blocking all-to-all P2P transfer capability of NVLink-based NVSwitch. RMG sort scales linearly with the number of GPUs  $g$  in the radix partitioning phase and keeps a constant P2P key swap time, independent of  $g$ .

**Sorting Large Data – DGX A100.** We compare the performance of the three multi-GPU sorts for increasing input sizes (up to 16B keys) on eight GPUs on the DGX A100. Compared to P2P merge sort, RMG sort is faster up to 1.3 $\times$  while outperforming HET merge sort up to 1.8 $\times$ . For 32B integer keys (128 GB), RMG sort takes about 3 seconds, which is 1.85 $\times$  faster than HET merge sort. Including the data transfers, RMG sort achieves an end-to-end sorting rate of over 10 billion keys per second and scales linearly to larger input sets.

### 5.3 Sorting Performance By Data Distributions and Data Types

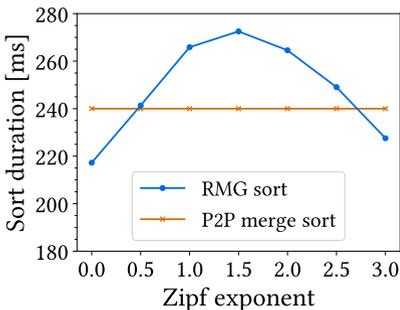
**Varying Data Distributions.** In Table 2, we compare the sorting time of RMG sort to that of P2P merge sort for varying distributions on the DGX A100 for eight GPUs. HET merge sort is stable across these distributions and purely bound by the main memory bandwidth. We sort 32-bit unsigned integers and find that RMG sort performs better on sorted (13%), nearly-sorted (8%), reverse-sorted (7%), and zero entropy (6%) distributions than for uniform ones. For the zero entropy distribution (i. e. all keys are the same), and already sorted data, RMG sort skips the P2P key swap. Nearly-sorted distributions require almost no key swaps. Additionally, for zero entropy data, we skip sorting the buckets as each one is a last-pass spanning bucket. During the partitioning, we computed the histograms  $p$  times, considered all  $k$  bits, but never scattered the keys. RMG sort is quick for read-intensive workloads. In the end, P2P merge sort and RMG sort are equally fast for zero and sorted data, while RMG sort is slightly faster on nearly-sorted keys. Reverse-sorted distributions benefit RMG sort as the keys are exchanged only between pairs of mirrored GPUs. We measure that this copy pattern achieves a higher P2P throughput (1.3 – 4.7 $\times$ ). Normal distributions require two partitioning passes. Overall, RMG sort outperforms P2P merge sort for reverse-sorted (30%), uniform (17%), and normal distributions (13%).

**Sorting Skewed Data.** In this section, we evaluate the performance of RMG sort for Zipfian distributions. For increasing Zipf exponents  $z$ , the probability of a key being one of only a

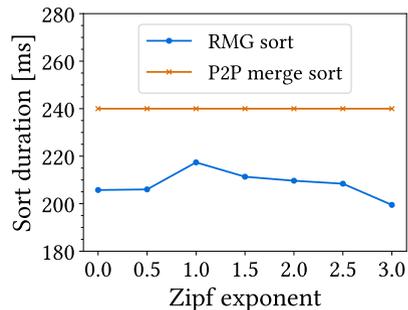
few highly frequent key values increases. Given two billion keys and  $z=1.0$ , the probability that a key is one of the top-1000 most occurring keys is 34%. The same probability is at 97.5% for  $z=1.5$ . In Figure 8a, we depict the sort duration of RMG sort for varying  $z$ . We sort 2B 32-bit keys on two GPUs on the IBM AC922. The sort duration increases for  $z > 0$ . At the peak at  $z=1.5$ , it reaches 273ms, (+26% over the sorting time of the uniform distribution). For  $z \geq 1.5$ , the sort duration steadily decreases down to the initial duration.

For  $z=0.5$ , one pass completes the partitioning phase. We measure average execution times for the key scattering and the histogram computation. However, the number of buckets on the second GPU is greater than our threshold  $MAX_{BRS} = 128$ , despite our optimization to fuse small neighbouring buckets. For more than  $MAX_{BRS}$  buckets, we sort the entire GPU chunk instead of individual buckets to avoid too many kernel launches. As a result, we cannot overlap the sorting with the DtoH copy, which explains why the total time increases by 30ms. For  $0.5 < z \leq 1.5$ , fusing neighbouring buckets reduces the total number of buckets significantly, e. g. from 575 to 90 in some cases. For  $z \geq 0.75$ , RMG sort performs the sort-copy-overlap at peak throughput and the bottleneck shifts to the radix partitioning.

For  $0.5 < z \leq 1.5$ , the distribution becomes more skewed, and more spanning buckets need to be resolved. For  $z=1.0$ , three partitioning passes are necessary, while the  $z=1.5$  requires all four. Thus, we have little to no sorting computation left after the partitioning phase considered (almost) all bits. However, the duration of multiple `ScatterKeys` kernel executions adds up significantly on this system, i. a. because of the many shared memory atomic conflicts. We measure the time of one `ScatterKeys` kernel execution to increase up to 20ms (+25%). This adds 40-70ms to the total sort duration for  $z \in [1.0, 1.5]$ . For  $z \geq 1.5$ , the sort duration decreases as almost all keys belong to the same few buckets, approaching the zero entropy distribution. Then, we increasingly skip the key scattering steps during the partitioning passes because all keys of a GPU chunk belong to the same bucket. Computing the histogram (6ms) takes less than the key scattering (16ms). Also, we do not sort the buckets since we partitioned on all  $k = 32$  bits. We evaluate the sort duration of P2P merge



(a) 2 GPUs on the IBM AC922



(b) 8 GPUs on the DGX A100

**Fig. 8: Sorting performance for skewed data (2 billion keys)**

sort and HET merge sort to be independent of the Zipf exponent. In the worst case of  $z=1.5$ , RMG sort is up to 13% slower than P2P merge sort, but 1.3× faster than HET merge sort.

In Figure 8b, we depict the sort duration of RMG sort for varying Zipf exponents  $z$  on the DGX A100 on eight GPUs. We measure significantly fewer differences in the sort duration for skewed data compared to the IBM AC922. The peak sort duration for  $z=1.0$  constitutes a 6% increase over the sorting time of uniform keys. Again, the time duration increases as the radix partitioning phase needs multiple passes. The low CPU-GPU interconnect bandwidth of the DGX A100 is one reason why the performance impact of the Zipf exponent is less significant on this system. Also, distributing the buckets across eight instead of two GPUs results in fewer buckets per GPU. Thus, the number of buckets per GPU is less likely to exceed  $MAX_{BRS}$ . For RMG sort, scaling up  $g$  reduces the negative performance impact of data skew. Moreover, the accumulated execution times for all histogram computations and key scattering steps are less than on the AC922 because 1) the NVIDIA A100 GPU has a higher global memory bandwidth and faster atomic operations, and 2) each GPU processes fewer keys. For skewed data on the eight GPUs of the DGX A100, RMG sort outperforms P2P merge sort by at least 11% and up to 1.3×, and HET merge sort by 1.7-1.8×.

**Sorting Different Data Types.** We evaluate RMG sort’s performance for unsigned integer and floating-point keys in their 32-bit and 64-bit variant. We sort unsigned key values, i. e. unsigned integer types, and positive floating-point numbers. Extending the algorithm to support negative value ranges is possible without sacrificing performance [Te00]. We sort uniformly distributed integers, and floating-point keys whose values follow a Zipfian distribution with an exponent of 1.0. In that way, the  $k$  bits of the floating-point keys are distributed similarly to the uniform integer distribution. When sorting 2 billion keys with two GPUs on the IBM AC922, the sort duration of 32-bit integers is approximately the same as for 32-bit floats. This is expected given that RMG sort’s time duration depends on the number of bits per key. However, on the IBM AC922, the sort duration of 64-bit data types is 2.2× higher than for the same number of 32-bit keys. This is the case for all three multi-GPU sorting algorithms, and confirms the findings of Maltenberger et al. [Ma22]. For the same experiment on the DGX A100, sorting 64-bit data types takes exactly 2× longer.

## 6 Related Work

Various single-GPU sorting algorithms have been proposed [Ba20, Ca17, DZ12, Go06, KW05, LOS10, SHG09, Sa10, SA08]. Ha et al. propose an LSB radix sort algorithm that considers two bits at a time and performs a block-local key shuffle in shared memory to ensure coalesced writes [HKS09]. Merrill et al. design an LSB radix sort algorithm that dynamically adjusts the number of keys a thread processes [MG11]. They also implement an analytical performance model that determines the optimal number of bits per pass, reducing the memory workload for any target architecture. Their approach has been integrated into NVIDIA’s high-performance CUB library [NV21a, Ad20]. Stehle et al. publish an MSB radix sort algorithm that increases the number of bits considered at a time to eight [SJ17].

They partition the keys into smaller and smaller buckets until a local sort algorithm sorts the buckets in on-chip memory. These sorting algorithms are single-GPU approaches. We publish a novel multi-GPU sorting algorithm. To the best of our knowledge, all previous multi-GPU sorting algorithms are merge-based. Peters et al. propose a multi-GPU sorting algorithm that sorts large-out-of-core data [PSHL10]. The authors use multiple GPUs to sort chunks. After copying the sorted chunks back to main memory, the CPU finds splitter elements to form disjoint data sets that individual GPUs can merge independently. Gowanlock et al. publish a heterogeneous multi-GPU sorting algorithm for large data where the CPU merges sorted chunks [GK18]. Both these algorithms sort large-out-of-core data, but neither one utilizes inter-GPU communication. Tanasic et al. propose a multi-GPU sorting algorithm for in-memory data that merges chunks using P2P transfers [Ta13].

## 7 Conclusion

In this paper, we design and evaluate the first radix-partitioning-based multi-GPU sorting algorithm (RMG sort). It outperforms CPU-only algorithms by up to 20×. Our MSB radix partitioning strategy exploits all-to-all P2P interconnects. As a result, RMG sort scales linearly with the input size and reduces the inter-GPU communication as it requires only one all-to-all P2P key swap, independent of  $g$ . Our evaluation shows that RMG sort utilizes high-speed P2P interconnects more efficiently than prior work. Compared to two state-of-the-art merge-based multi-GPU sorting algorithms, RMG sort scales best with increasing numbers of GPUs. We outperform P2P merge sort up to 1.3× and HET merge sort up to 1.8×. When we exclude the CPU-GPU copy times to directly compare the on-GPU computation and P2P communication, RMG sort is 2.7× faster than P2P merge sort and 9.2× faster than HET merge sort. Thus, RMG sort benefits from future accelerator platforms given that hardware vendors continue to increase the number of GPUs, and the P2P and CPU-GPU interconnect bandwidth [NV22c, NV21c, NV22d].

RMG sort can improve the performance of an existing out-of-core sort-merge algorithm, reducing the number of chunks that need to be merged. Alternatively, we suggest extending RMG sort by a preliminary partitioning step on the CPU which divides the input into chunks of distinct value ranges that fit into the combined GPU memory. Then, out-of-core data sets are sorted in independent sorting rounds. In future work, we want to analyze RMG sort's performance as part of a real-world database use case. RMG sort can be extended to support key-value pairs. For each key permutation, we use the block-local histograms to rearrange the corresponding values equivalently in the key scattering step.

### Acknowledgments

The authors would like to thank Elias Stehle for his input throughout the algorithm design. This work was partially funded by the German Ministry for Education and Research (ref. 01IS18025A and ref. 01IS18037A), the German Research Foundation (ref. 414984028), and the European Union's Horizon 2020 research and innovation program (ref. 957407).

## Bibliography

- [Ad20] Adinets, A.: , A Faster Radix Sort Implementation. <https://developer.download.nvidia.com/video/gputechconf/gtc/2020/presentations/s21572-a-faster-radix-sort-implementation.pdf>, October 2020. Last accessed: 2022-04-26.
- [Ag96] Agarwal, Ramesh C.: A Super Scalar Sort Algorithm for RISC Processors. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, p. 240–246, 1996.
- [AM18] AMD: , AMD Radeon Instinct MI60: Unleash Discovery on the World’s Fastest Double Precision PCIe Accelerator. <https://www.amd.com/system/files/documents/radeon-instinct-mi60-datasheet.pdf>, November 2018. Last accessed: 2022-04-26.
- [AM20] AMD: , Introducing AMD CDNA Architecture: The All-New AMD GPU Architecture for the Modern Era of HPC and AI (Whitepaper). <https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf>, December 2020. Last accessed: 2022-04-26.
- [Ba20] Baxter, Sean: , Modern GPU: Patterns and Behaviors for GPU Computing. <https://github.com/moderngpu/moderngpu>, January 2020. Last accessed: 2022-04-26.
- [Ca17] Casanova, Henri; Iacono, John; Karsin, Ben; Sitchinava, Nodari; Weichert, Volker: An Efficient Multiway Mergesort for GPU Architectures. Technical report, arXiv:cs.DS, February 2017.
- [Ch15] Cho, M.; Brand, D.; Bordawekar, R.; Finkler, U.; Kulandaisamy, V.; Puri, R.: PARADIS: An Efficient Parallel Algorithm for In-Place Radix Sort. Proc. VLDB Endow., 8(12):1518–1529, August 2015.
- [CI18] Colgan, Maria; Insider, Oracle Database: , Does GPU Hardware Help Database Workloads? <https://blogs.oracle.com/database/post/does-gpu-hardware-help-database-workloads>, February 2018. Last accessed: 2022-04-26.
- [DZ12] Dehne, Frank; Zaboli, Hamidreza: Deterministic Sample Sort for GPUs. Parallel Processing Letters, 22(3):1–14, September 2012.
- [FS21] FSF: , The GNU C++ Library Manual: Parallel Mode. [https://gcc.gnu.org/onlinedocs/gcc-11.2.0/libstdc++/manual/manual/parallel\\_mode.html](https://gcc.gnu.org/onlinedocs/gcc-11.2.0/libstdc++/manual/manual/parallel_mode.html), July 2021. Last accessed: 2022-04-26.
- [Gi19] Gill, Sandeep Kaur; Singh, Virendra Pal; Sharma, Pankaj; Kumar, Durgesh: A Comparative Study of Various Sorting Algorithms. International Journal of Advanced Studies of Scientific Research, 4(1), February 2019.
- [GK18] Gowanlock, Michael; Karsin, Ben: Sorting Large Datasets with Heterogeneous CPU/GPU Architectures. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Institute of Electrical and Electronics Engineers, pp. 560–569, 2018.
- [Go06] Govindaraju, Naga; Gray, Jim; Kumar, Ritesh; Manocha, Dinesh: GPUteraSort: High Performance Graphics Co-Processor Sorting for Large Database Management. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. SIGMOD ’06. Association for Computing Machinery, p. 325–336, 2006.

- [Gr06] Graefe, Goetz: Implementing Sorting in Database Systems. *ACM Comput. Surv.*, 38(3):1–37, September 2006.
- [Gu15] Gupta, Anurag; Agarwal, Deepak; Tan, Derek; Kulesza, Jakub; Pathak, Rahul; Stefani, Stefano; Srinivasan, Vidhya: Amazon Redshift and the Case for Simpler Data Warehouses. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, p. 1917–1923, 2015.
- [HKS09] Ha, Linh; Krueger, Jens; Silva, Claudio T.: Fast Four-Way Parallel Radix Sorting on GPUs. *Computer Graphics Forum*, 2009.
- [Ja14] Jagadish, H. V.; Gehrke, Johannes; Labrinidis, Alexandros; Papakonstantinou, Yannis; Patel, Jignesh M.; Ramakrishnan, Raghu; Shahabi, Cyrus: Big Data and Its Technical Challenges. *Commun. ACM*, 57(7):86–94, July 2014.
- [KW05] Kipfer, Peter; Westermann, Rüdiger: , Chapter 46. Improved GPU Sorting. <https://developer.nvidia.com/gpugems/gpugems2/part-vi-simulation-and-numerical-algorithms/chapter-46-improved-gpu-sorting>, April 2005. Last accessed: 2022-04-26.
- [Li20] Li, Ang; Song, Shuaiwen Leon; Chen, Jieyang; Li, Jiajia; Liu, Xu; Tallent, Nathan R.; Barker, Kevin J.: Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 31(1):94–110, January 2020.
- [LOS10] Leischner, Nikolaj; Osipov, Vitaly; Sanders, Peter: GPU Sample Sort. In: *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. pp. 1–10, 2010.
- [Lu20] Lutz, Clemens; Breß, Sebastian; Zeuch, Steffen; Rabl, Tilmann; Markl, Volker: Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, pp. 1633–1649, 2020.
- [Ma22] Maltenberger, Tobias; Ilic, Ivan; Tolovski, Ilin; Rabl, Tilmann: Evaluating Multi-GPU Sorting with Modern Interconnects. In: *2022 ACM SIGMOD International Conference on Management of Data (SIGMOD '22)*. Association for Computing Machinery, 2022.
- [MG11] Merrill, Duane; Grimshaw, Andrew: High Performance and Scalable Radix Sorting: A Case Study of Implementing Dynamic Parallelism for GPU Computing. *Parallel Processing Letters*, 21(2):245–272, June 2011.
- [MG16] Merrill, Duane; Garland, Michael: Single-Pass Parallel Prefix Scan with Decoupled Look-Back. Technical report, NVIDIA, August 2016. [https://research.nvidia.com/sites/default/files/pubs/2016-03\\_Single-pass-Parallel-Prefix/nvr-2016-002.pdf](https://research.nvidia.com/sites/default/files/pubs/2016-03_Single-pass-Parallel-Prefix/nvr-2016-002.pdf).
- [NSH18] Nohria, Ritesh; Santos, Gustavo; Haug, Volker: , IBM Power System AC922: Technical Overview and Introduction. <https://www.redbooks.ibm.com/redpapers/pdfs/redp5494.pdf>, July 2018. Last accessed: 2022-04-26.
- [NV17] NVIDIA: , NVIDIA Tesla V100 GPU Architecture. <http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf>, August 2017. Last accessed: 2022-04-26.

- [NV18] NVIDIA: , Technical Overview NVIDIA NVSwitch: The World's Highest-Bandwidth On-Node Switch. <http://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf>, April 2018. Last accessed: 2022-04-26.
- [NV20] NVIDIA: , NVIDIA A100 Tensor Core GPU Architecture. <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf>, September 2020. Last accessed: 2022-04-26.
- [NV21a] NVIDIA: , CUB: Cooperative Primitives for CUDA C++. <https://github.com/NVIDIA/cub>, June 2021. Last accessed: 2022-04-26.
- [NV21b] NVIDIA: , DGX A100 System User Guide. <https://docs.nvidia.com/dgx/pdf/dgxa100-user-guide.pdf>, November 2021. Last accessed: 2022-04-26.
- [NV21c] NVIDIA: , NVIDIA Grace CPU. <https://www.nvidia.com/en-us/data-center/grace-cpu/>, April 2021. Last accessed: 2022-04-26.
- [NV21d] NVIDIA: , Thrust: Code at the Speed of Light. <https://github.com/NVIDIA/thrust>, June 2021. Last accessed: 2022-04-26.
- [NV22a] NVIDIA: , CUDA C++ Best Practices Guide. [https://docs.nvidia.com/cuda/pdf/CUDA\\_C\\_Best\\_Practices\\_Guide.pdf](https://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf), January 2022. Last accessed: 2022-04-26.
- [NV22b] NVIDIA: , CUDA C++ Programming Guide. [https://docs.nvidia.com/cuda/pdf/CUDA\\_C\\_Programming\\_Guide.pdf](https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf), January 2022. Last accessed: 2022-04-26.
- [NV22c] NVIDIA: , NVIDIA DGX H100: The Gold Standard for AI Infrastructure. <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-dgx-h100-datasheet.pdf>, March 2022. Last accessed: 2022-04-26.
- [NV22d] NVIDIA: , NVIDIA HGX AI Supercomputer: The most powerful end-to-end AI supercomputing platform. <https://www.nvidia.com/en-us/data-center/hgx/>, January 2022. Last accessed: 2022-04-26.
- [Pa21] Paul, Johns; Lu, Shengliang; He, Bingsheng; Lau, Chiew Tong: MG-Join: A Scalable Join for Massively Parallel Multi-GPU Architectures. In: Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data. Association for Computing Machinery, pp. 1413–1425, 2021.
- [Pe19] Pearson, C.; Dakkak, A.; Hashash, S.; Li, C.; Chung, I-H.; Xiong, J.; Hwu, W.-M.: Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects. In: Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering. ICPE '19. Association for Computing Machinery, pp. 209–218, 2019.
- [PSHL10] Peters, Hagen; Schulz-Hildebrandt, Ole; Luttenberger, Norbert: Parallel External Sorting for CUDA-Enabled GPUs with Load Balancing and Low Transfer Overhead. In: 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW). Institute of Electrical and Electronics Engineers, pp. 1–8, 2010.
- [Ra20] Raza, Aunn; Chrysogelos, Periklis; Sioulas, Panagiotis; Indjic, Vladimir; Anadiotis, Angelos Christos; Ailamaki, Anastasia: GPU-Accelerated Data Management under the Test of Time. Online proceedings of the 10th Conference on Innovative Data Systems Research (CIDR), pp. 1–11, 2020.

- [RLT20] Rui, Ran; Li, Hao; Tu, Yi-Cheng: Efficient Join Algorithms for Large Database Tables in a Multi-GPU Environment. *Proc. VLDB Endow.*, 14(4):708–720, December 2020.
- [SA08] Sintorn, Erik; Assarsson, Ulf: Fast Parallel GPU-Sorting Using a Hybrid Algorithm. *Journal of Parallel and Distributed Computing*, 68(10):1381–1388, October 2008.
- [Sa10] Satish, Nadathur; Kim, Changkyu; Chhugani, Jatin; Nguyen, Anthony D.; Lee, Victor W.; Kim, Daehyun; Dubey, Pradeep: Fast Sort on CPUs and GPUs: A Case for Bandwidth Oblivious SIMD Sort. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, pp. 351–362, 2010.
- [SHG09] Satish, Nadathur; Harris, Mark; Garland, Michael: Designing Efficient Sorting Algorithms for Manycore GPUs. In: *2009 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. Institute of Electrical and Electronics Engineers, pp. 1–10, 2009.
- [SJ17] Stehle, Elias; Jacobsen, Hans-Arno: A Memory Bandwidth-Efficient Hybrid Radix Sort on GPUs. In: *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, pp. 417–432, 2017.
- [SMY20] Shanbhag, Anil; Madden, Samuel; Yu, Xiangyao: A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics (Extended Version). Technical report, Massachusetts Institute of Technology, March 2020.
- [ST20] Sharma, Debendra Das; Tavallaei, Siamak: , Compute Express Link 2.0 White Paper. [https://b373eaf2-67af-4a29-b28c-3aae9e644f30.filesusr.com/ugd/0c1418\\_14c5283e7f3e40f9b2955c7d0f60bebe.pdf](https://b373eaf2-67af-4a29-b28c-3aae9e644f30.filesusr.com/ugd/0c1418_14c5283e7f3e40f9b2955c7d0f60bebe.pdf), November 2020. Last accessed: 2022-04-26.
- [Ta13] Tanasic, Ivan; Vilanova, Lluís; Jordà, Marc; Cabezas, Javier; Gelado, Isaac; Navarro, Nacho; Hwu, Wen-mei: Comparison Based Sorting for Systems with Multiple GPUs. In: *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*. Association for Computing Machinery, pp. 1–11, 2013.
- [Te00] Terdiman, Pierre: , Radix Sort Revisited. <http://www.codercorner.com/RadixSortRevisited.htm>, January 2000. Last accessed: 2022-04-26.
- [ZB91] Zagha, Marco; Blelloch, Guy E.: Radix Sort for Vector Multiprocessors. In: *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*. Supercomputing '91. Association for Computing Machinery, p. 712–721, 1991.
- [ZW12] Zhang, Keliang; Wu, Baifeng: A Novel Parallel Approach of Radix Sort with Bucket Partition Preprocess. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication, 2012 IEEE 9th International Conference on Embedded Software and Systems*. pp. 989–994, 2012.

# Approach to Synthetic Data Generation for Imbalanced Multi-class Problems with Heterogeneous Groups

Dennis Treder-Tschechlov,<sup>1</sup> Peter Reimann,<sup>2</sup> Holger Schwarz,<sup>1</sup> Bernhard Mitschang<sup>1</sup>



**Abstract:** To benchmark novel classification algorithms, these algorithms should be evaluated on data with characteristics that also appear in real-world use cases. Important data characteristics that often lead to challenges for classification approaches are multi-class imbalance and heterogeneous groups. Heterogeneous groups are sets of real-world entities, where the classification patterns may vary among different groups and where the groups are typically imbalanced in the data. Real-world data that comprise these characteristics are usually not publicly available, e. g., because they constitute sensitive patient information or due to privacy concerns. Further, the manifestations of the characteristics cannot be controlled specifically on real-world data. A more rigorous approach is to synthetically generate data such that different manifestations of the characteristics can be controlled as well. However, existing data generators are not able to generate data that feature both data characteristics, i. e., multi-class imbalance and heterogeneous groups. In this paper, we propose an approach that fills this gap as it allows to synthetically generate data that exhibit both characteristics. We make use of a taxonomy model that organizes real-world entities in domain-specific heterogeneous groups to generate data reflecting the characteristics of these groups. Further, we incorporate probability distributions to reflect the imbalances of multiple classes and groups from real-world use cases. The evaluation shows that our approach can generate data that feature the data characteristics multi-class imbalance and heterogeneous groups and that it allows to control different manifestations of these characteristics.

**Keywords:** Machine Learning, Classification, Data Generation, Real-world Data Characteristics

## 1 Introduction

Data are the basis to evaluate and benchmark classification algorithms. For such benchmarks, algorithms should be evaluated on data that reflect characteristics that also appear in real-world use cases. Besides general characteristics, such as the number of data instances, features, or classes, we focus on characteristics that lead to significant challenges for classification algorithms and that are present in many real-world use cases. According to major literature in this field, two of the most challenging data characteristics are **multi-class imbalance** [HG09, Ga12, WY12] and **heterogeneous groups** [SWK09, HRM19, Me21, SG21]. Multi-class imbalance means that the data contains multiple classes that are unevenly distributed [HG09]. This leads to the challenge that less frequent classes are typically ignored by classification algorithms [WY12]. We define groups as specific sets of real-world entities, e. g., genders, ethnic groups [SG21], or product groups [HRM19]. These groups are usually heterogeneous

<sup>1</sup> University of Stuttgart, IPVS, 70569 Stuttgart, Germany, {firstname.lastname}@ipvs.uni-stuttgart.de

<sup>2</sup> University of Stuttgart, GSaME, 70569 Stuttgart, Germany, peter.reimann@gsame.uni-stuttgart.de

in real-world data in the sense that classification patterns may vary among the groups and that the groups are imbalanced in the data [HRM19, Me21, SG21]. This leads to very different analysis results when performing the same analysis on the entire data or on each group separately [Me21, SG21]. Both characteristics together are present in several use cases, e. g., in use cases across the industrial value chain [KBT11, Su14, Wu16, KM16, HRM19, Gr22] or in medical use cases [KU15, SGG18, Ch20, MGTM20, SG21].

However, it is typically not possible to obtain representative benchmark data that contain both a multi-class imbalance and heterogeneous groups. Real-world datasets are usually not publicly available, e. g., because they constitute sensitive patient information or due to privacy concerns. Obtaining data from publicly available repositories, e. g., OpenML [Va14], is also not a feasible option, since they do not reflect both characteristics to the same extent as they occur in data of real-world use cases. To make meaningful evaluations and benchmarks of classification algorithms, it should furthermore be possible to control the manifestations of both characteristics in the data, i. e., the degree of class imbalance or the heterogeneity and imbalance of domain-specific groups. This is however neither possible with individual real-world datasets nor with data from common repositories. Therefore, a more rigorous approach is to generate synthetic data with both multi-class imbalance and heterogeneous groups. Existing data generators (e. g., [SH05, Fr11, FS18, Ig19, Gu03]) are able to generate data characteristics that are based on statistical properties. That is, they can generate data with different degrees of multi-class imbalance. Yet, to generate data with heterogeneous groups, domain knowledge about the groups from real-world application domains is required. However, existing data generators do not use such domain knowledge and thus are not able to generate data with heterogeneous groups. In addition, when the data to be generated has to contain both a multi-class imbalance and heterogeneous groups, a data generator has to ensure that all dependencies between both data characteristics are properly reflected within the data. For instance, the class imbalance not only has to be reflected within the whole dataset, but also within each subset of the respective groups.

In this paper, we propose an approach to generate data synthetically that mitigates the drawbacks of existing data generators and publicly available data repositories. That is, our approach is capable of generating numerical and categorical data with both multi-class imbalance and heterogeneous groups. An important design guideline of our approach is the applicability in many different domains. Our contributions include the following:

- Our approach is the first that is capable of generating numerical and categorical data with domain-specific heterogeneous groups. To realize this, we use a taxonomy that organizes such groups in a hierarchical structure. A taxonomy is the simplest form of knowledge models and can thus be found in a wide range of domains. In consequence, our approach can be used in a variety of domains as well.
- We use probability distributions to reflect the imbalances of real-world groups and classes. To this end, we state requirements that such probability distributions have to fulfill. In our approach, we use the Zipf distribution as it fulfills all requirements.

- We propose a two-step procedure to generate the data synthetically based on the taxonomy and the probability distribution. First, we traverse the taxonomy top-down to specify important data characteristics regarding the group distribution. Second, we specify the class distributions among the groups and generate the data bottom-up.
- In our evaluation, we show that the data our approach may generate comprise both data characteristics. To this end, we assess the characteristics with different metrics such as classification complexity, statistics, or a detailed view on classification accuracy. In addition, we show to which extent individual parameters of our approach influence the characteristics and the aforementioned metrics.

The rest of this paper is structured as follows: In Section 2, we define the characteristics that we generate with our proposed approach. We examine limitations of existing data repositories and data generators in Section 3. In Section 4, we describe our method to generate data synthetically and how this method may be parameterized to incorporate different degrees of each characteristic. We discuss the results of evaluating our data generation approach with different parameter configurations in Section 5. Section 6 finally concludes our work.

## 2 Data Characteristics

We assume a classification problem with  $c > 2$  class labels, i. e., a multi-class problem with classes  $C = \{C_1, \dots, C_c\}$ . This problem consists of a dataset  $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^n$  with  $n$  tuples, where each tuple contains an instance  $x_i$  of the data and a class label  $y_i \in C$ . Here,  $x_i$  is a feature-vector from a  $f$ -dimensional feature space  $\mathcal{F} = \{F_1, \dots, F_f\}$ . Each feature  $F_i$  either has categorical or numerical values. The problem is then to generate data that comprise the data characteristics multi-class imbalance (DC1) and heterogeneous groups (DC2).

### 2.1 Multi-Class Imbalance (DC1)

A prevalent challenge in literature and in many practical scenarios is *multi-class imbalance* [HG09, WY12, Wu16]. Here, certain classes are represented more often in  $\mathcal{X}$  than other classes. The classes occurring more often  $C^+ \subset C$  are called majority classes, while the less frequent classes  $C^- = C \setminus C^+$  are called minority classes. Machine learning algorithms aim to optimize the overall accuracy of the predictions for the whole dataset  $\mathcal{X}$ . Majority classes have a big share of all instances in  $\mathcal{X}$  so that the overall accuracy highly correlates with the accuracy of predictions for these majority classes. Thus, machine learning algorithms tend to ignore the instances  $\mathcal{X}^-$  of minority classes and are therefore biased towards majority classes. However, making accurate predictions for minority classes is in many real-world use cases even more important [Wu16]. In data-driven medical diagnoses, they may represent rare, but dangerous or even lethal diseases that must be detected with the highest accuracy possible [Ch20]. While literature comprises many approaches that may handle binary

class imbalance, e. g., approaches to over- or under-sampling, these approaches are not able to deal with multi-class imbalance [WY12, Fe13]. Further, multi-class imbalance often comprises accompanying symptoms that likewise lead to challenges for classification algorithms [HG09, WY12]. For instance, this concerns overlapping classes, i. e., instances from classes that are next to each other (a.k.a border points [HB02]).

## 2.2 Heterogeneous Groups (DC2)

Data in real-world scenarios often represent the observations for diverse groups of entities. In industrial use cases, these groups of entities constitute the various product groups, e. g., a vehicle engine can be differentiated by 'Diesel' or 'Gasoline' engines, which each may be further divided into four- and six-cylinder engines [HRM20]. In medical applications, different groups of patient populations exist, e. g., patients with different gender types [ULP19, WLL21] or with different skin colors [Ch20]. Hence, data from real-world use cases typically comprise  $k$  domain-specific groups  $G_1, \dots, G_k \subset \mathcal{X}$  such that  $\bigcup_{i=1}^k G_i = \mathcal{X}$  and  $G_i \cap G_j = \emptyset$  for  $i \neq j$ . Further, each group  $G_i$  can comprise multiple classes  $C_{i1}, \dots, C_{ic}$ , while each class may be included in multiple groups. These real-world groups and their data are heterogeneous in the sense that they are distributed in an imbalanced way in the dataset  $\mathcal{X}$  (DC2a) and show heterogeneous class patterns (DC2b).

**Imbalanced Groups (DC2a):** Data from real-world use cases often comprise imbalanced groups, i. e., some groups occur more frequently than others. In medical applications, different groups of patients are typically more frequent than others, e. g., patients with lighter skin colors are typically more frequent than patients with darker skin colors [Ch20]. For industrial use cases, certain product groups occur more often than others [Su14, Wu16, KM16, HRM20], e. g., 'Gasoline' engines are more frequent than 'Diesel' engines. Thus, the dataset  $\mathcal{X}$  comprises majority groups  $G^+ \subset \mathcal{X}$  that appear more frequently than minority groups  $G^- = \mathcal{X} \setminus G^+$ . Learning algorithms tend to favor the majority groups  $G^+$  as these comprise much more instances of the data. Further, the data may comprise underrepresented minority groups  $G^-$  that occur very rarely and may thus be ignored by classification algorithms. This is also known as representation bias in literature [Me21, SG21].

**Heterogeneous Class Patterns (DC2b):** Real-world use cases often exhibit a heterogeneity of class patterns, i. e., a single class is described by different patterns in the data subsets of different groups  $G_i$ . An example in medical applications is that the symptoms for specific types of skin cancer vary for different skin colors [Ch20], i. e., the different groups (skin colors) exhibit different patterns (symptoms) for the same class (cancer type). In many industrial use cases, the patterns for the same class likewise vary across different product groups [Su14, KM16, Wu16, HRM20, Wi20]. In different groups, the same class may for instance have different value ranges for the same feature [HRM20]. This heterogeneity of class patterns usually leads to an aggregation bias [Me21, SG21]. That is, a particular data analysis carried out on the entire dataset  $\mathcal{X}$  yields different results than the very same analysis performed on each group  $G_i$  separately.

Tab. 1: Overview of related repositories and data generators regarding their data characteristics and domain-independence. A '✓' means that the characteristic can be generated or that the criterion is fulfilled, while a '✗' means the opposite.

| Category                        | Examples  | Domain-Independent | DC1: Multi-class Imbalance | DC2: Heterogeneous Groups |
|---------------------------------|---|--------------------|----------------------------|---------------------------|
| Repositories                    | UCI [DG17], OpenML [Va14], KEEL [A111], Kaggle, etc.                          | ✗                  | ✓                          | ✗                         |
| Domain-specific Data Generators | fraud detection [LKJ02], health care [DC19], production-oriented [Fe20], etc. | ✗                  | ✓                          | ✗                         |
| Data Augmentation               | GANs [GBC16, RHW21], SDV [PWV16], etc.  | ✗                  | ✓                          | ✗                         |
| Domain-agnostic Data Generators | Clustering [SH05, Fr11, FS18, Ig19], Classification [Gu03], Scikit-learn      | ✓                  | ✓                          | ✗                         |

### 3 Related Work

Table 1 summarizes our key findings of related work, which we discuss in the following.

**Machine Learning Repositories:** Literature often makes use of data from publicly available machine learning repositories to develop and evaluate novel machine learning algorithms. Several machine learning repositories exist, e. g., OpenML [Va14], KEEL [A111], Kaggle<sup>3</sup>, and the UCI ML Repository [DG17]. These repositories include around 3500 datasets. The above mentioned repositories together only offer 40 data sets with more than 10 classes and with at least a moderate multi-class imbalance (DC1). Regarding heterogeneous groups (DC2), some works in the area of fair machine learning consider the adult<sup>4</sup> or COMPASS<sup>5</sup> dataset, where different groups of gender or skin color are present, e. g., 'male' and 'female' or 'white' and 'black' [ULP19, WLL21]. Yet, these datasets typically contain only two or up to four groups [ULP19, WLL21]. For industrial use cases, there may be thirty [HRM20] or even thousands [Su14] of different groups in the data. Hence, data from publicly available repositories do not have heterogeneous class patterns or imbalanced groups (DC2) to the same extent as found in real-world use cases [Su14, HRM20, Ch20]. Further, each dataset is specific for a certain domain and thus we evaluate the repositories as domain-dependent.

**Domain-specific Data Generators:** As common repositories do not offer data containing both data characteristics, the next possibility is to generate data synthetically. Literature comprises different synthetic data generators that focus on specific domains, e. g., fraud detection [LKJ02], health care applications [DC19], or production-oriented data [Fe20]. These works are typically structured into two steps: First, they define a specific data model for the domain at hand and second, generate the data according to this data model. Yet,

<sup>3</sup> Kaggle datasets: <https://www.kaggle.com/datasets>

<sup>4</sup> Adult dataset: <https://archive.ics.uci.edu/ml/datasets/adult>

<sup>5</sup> Machine Bias article: <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>

these data models are specific to the given domain. For instance, Fernandes et al. pre-define 15 production-oriented features for the data model such as customer demand, total parts, or available time [Fe20]. Hence, these generators are only applicable to the domains for which appropriate data models are available [SB21]. Further, none of them focus on DC2. Although they use domain knowledge in terms of the data model, this data model usually does not consider the domain-specific groups or their heterogeneity. Therefore, they are for instance not able to generate different class patterns for these groups (see Table 1).

**Data Augmentation from Real-World Data:** The next group of synthetic data generators requires existing real-world data and uses this data as basis for data augmentation [RHW21, PWV16]. These approaches take a sample of the existing data, learn the distribution of the data sample and then generate new data from the learned distributions [RHW21, PWV16]. They typically employ generative adversarial networks (GANs) [GBC16], which are based on two networks: A generative network learns to map a latent space to a data distribution, and a discriminative network then draws samples from the learned distribution. However, these approaches require existing real-world data to model the distribution of DC1 and DC2. Since available data only covers DC1 (see Table 1), approaches to data augmentation can only generate new data that resembles an existing multi-class imbalance. Yet, they are not able to generate heterogeneous groups (DC2).

**Domain-Agnostic Data Generators:** The last group of related work covers domain-agnostic data generators [SB21], i. e., generators that are independent of specific domains and that do not require real-world data as basis. Literature comprises domain-agnostic generators for clustering [SH05, Fr11, FS18, Ig19] or for classification tasks [Gu03]. The approaches can vary the number of instances  $n$ , the number of features  $f$ , and the number of classes  $c$ . Most of them are able to generate data with multi-class imbalance (DC1). For example, scikit-learn<sup>6</sup> offers a data generator for classification that is taken from Guyon [Gu03]. To generate multi-class imbalance, users can pass in weights  $w = \{w_1, \dots, w_c\}$  for each class. These weights define the share of instances for each class, i. e.,  $w_i \in [0; 1]$  and  $\sum_{i=1}^c w_i = 1$ . The approach then generates the instances of each class separately. To this end, it generates for each class one or several clusters, where each class  $c_i$  has  $n \cdot w_i$  instances. It can generate multiple clusters for each class, but each cluster has the same number of instances. Hence, the domain-agnostic generators do not generate clusters (or groups) in the data with varying cluster sizes, i. e., imbalanced groups (DC2a). Moreover, they do not generate a heterogeneity of the class patterns among different clusters (DC2b). The reason is that they do not use domain knowledge about the groups and their distributions.

Summarizing related work, none of the related approaches is able to generate datasets that show both data characteristics (see Table 1), i. e., multi-class imbalance (DC1) and heterogeneous groups (DC2). Nevertheless, our approach employs existing domain-agnostic data generators to generate data with DC1 and extends them to also generate data with DC2, as well as with different manifestations of both DC1 and DC2.

<sup>6</sup> Scikit-learn data generator: [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make\\_classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification)

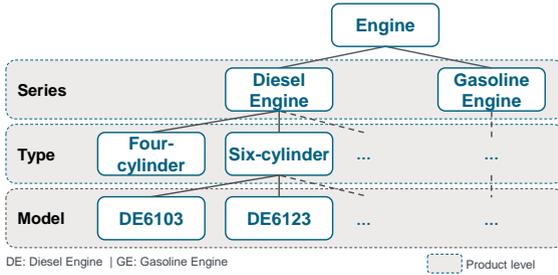


Fig. 1: Simplified excerpt of a taxonomy that defines hierarchical relationships of various engine types.

## 4 Approach to Synthetic Data Generation of DC1 and DC2

In this section, we describe our approach to generate data synthetically that comply with both data characteristics DC1 and DC2. Our approach can vary the number of instances  $n$ , features  $f$ , and classes  $c$ . Furthermore, our approach has the following parameters that influence the manifestations of DC1 and DC2:

- $s_C \in \mathbb{R}_{\geq 0}$ : Parameter to control the extent of imbalance for the classes in the generated data (DC1).
- $s_G \in \mathbb{R}_{\geq 0}$ : Parameter to control the extent of imbalance for the groups (DC2a).
- $co \in \mathbb{R}_{\geq 1}$ : Controls the class overlap between the groups. For  $co = 1$ , the classes are distributed disjoint across the groups, i. e., each class occurs only in one group. The higher the value, the more classes are in one group und thus finally also more classes tend to overlap across the groups. Then, the classification task gets more difficult.
- $gs \in \mathbb{R}_{\geq 0}$ : Parameter to control the group separation. It mainly influences the heterogeneity of the groups (DC2b). A low value means that the groups highly overlap with respect to the feature ranges of their instances. Higher values indicate more clearly separable groups.
- $cf \in \{1, 2, \dots, f\}$ : Number of characteristic features for each group, i. e., the number of features for which we separate the groups with  $gs$  (DC2b).

In the following sections, we first describe the domain knowledge model that we use to generate data that comprise real-world groups (DC2b). Subsequently, we examine probability distributions to generate data with imbalanced distributions for the classes (DC1) and for groups (DC2a). Finally, we detail on our algorithms to generate the data synthetically.

## 4.1 Taxonomy

For the generation of data with the characteristic DC2, we use information and characteristics regarding groups occurring in an application domain. Usual ways to model domain-specific groups and their relationships are knowledge graphs [Ho21] and semantic nets, e. g., taxonomies, thesauruses, or ontologies [So91]. Such models commonly organize the entities of a domain, amongst others, via hierarchical relationships of superordinate and subordinate groups. Subordinate groups, i. e., child nodes in the hierarchy, are more specific than their associated superordinate groups, i. e., the parent nodes [So91]. Hence, data related to any subordinate child group is a subset of the data of its parent group.

Our approach to generate data solely builds on hierarchical relationships among domain-specific groups. We do not need other more complex types of relationships as found in thesauruses, ontologies, or knowledge graphs. Hence, it is sufficient to use taxonomies, which already come with hierarchical relationships. As taxonomies are the simplest form of a semantic net, the effort to create them is moderate. Hence, they are pre-defined in various domains, e. g., for product families [AK04, Su14, HRM20] or skin colors of patients [Ja04]. Even if not present, literature comprises several ontology learning approaches to extract hierarchies from data, e. g., hierarchical clustering or association rule discovery [Ci09].

We define the hierarchical tree structure of a taxonomy as  $T = (V, E)$  with nodes  $V = \{v_1, \dots, v_l\}$  and edges  $E \subseteq V \times V$ .  $T$  is a directed, acyclic tree with exactly one root node and where each child node has exactly one parent node. An edge  $E_{ij} = (v_i, v_j)$  furthermore represents the hierarchical relationship of nodes  $v_i$  and  $v_j$ , i. e.,  $v_i$  is the parent node or the superordinate concept of  $v_j$ . This taxonomy model is also often encoded in the data itself. Therefore, the levels of  $T$  are encoded by level-specific key features. A node  $v_i$  on a level  $l$  has a distinct value in the key feature specific to level  $l$ .

**Example:** Figure 1 shows an example of a taxonomy that defines the hierarchical relationships of different engine types of motor vehicles. This taxonomy is adapted from a recent work regarding the end-of-line testing of complex truck engines [HRM20]. In this example, an engine is first distinguished between Diesel or Gasoline engines, and further divided by its engine type and model. Thus, we have three level-specific key features *series*, *type*, and *model*. The level-specific feature *series* may have one of the distinct values 'Diesel Engine' or 'Gasoline Engine'. Furthermore, every model 'DE6123' has also the type 'Six-cylinder', and is of series 'Diesel Engine', i. e., the edges describe the hierarchical relationships.

## 4.2 Probability Distribution

In our data generation, we use probability distributions ( $PD$ ) to reflect the imbalances of the classes (DC1) and groups (DC2a). That is, we use  $PD$  in two ways: First, to assign the number of instances for each class, and second, to assign the number of instances and classes among the groups. Therefore, the probability distribution has to (i) sample *univariate*

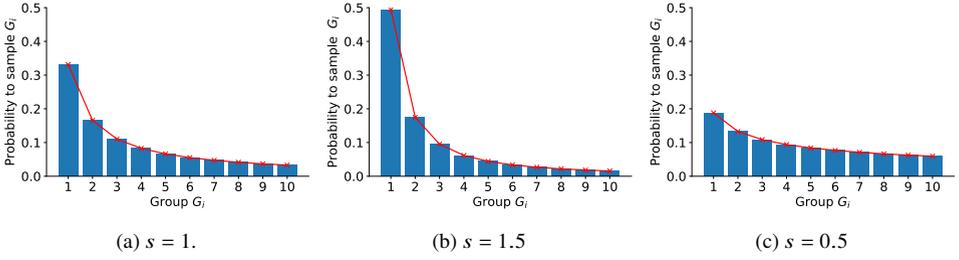


Fig. 2: Probability distribution to sample a group from 10 possible groups with the Zipf distribution. We vary the parameter  $s$  with (a) default value  $s = 1$ , (b) a more imbalanced setting with  $s = 1.5$ , and (c) a more balanced setting with  $s = 0.5$ .

random variables, (ii) reflect *imbalanced distributions* of the groups or classes, (iii) be *discrete*, as we sample integer values, i. e., the number of instances for the groups or classes, (iv) allow for *specifying the number of possible values* to sample from, e. g., we want to sample from 10 groups or 100 classes, and (v) allow to *parameterize* the distribution such that we are able to generate different manifestations of DC1 and DC2a.

A distribution that fulfills these criteria is the Zipf distribution from the family of exponential or power-law distributions [Sc12]. We examined several distributions that fulfill our criteria, e. g., Boltzman and Poisson [Sc12], but Zipf obtained the most similar data distribution compared to real-world data [HRM19,HRM20]. The first input is the number of classes  $c$  to generate. Furthermore, we have information about the groups from the taxonomy, i. e., we assume we have  $k$  groups  $G_1, G_2, \dots, G_k$  that are the leaf nodes of the taxonomy. We also assume that a ranking exists that orders the groups by the number of samples  $|G_i|$  to generate for each group  $G_i$ , i. e.,  $|G_1| \geq |G_2| \geq \dots \geq |G_k|$ . Hence, given the number of groups  $k$ , the rank  $r_i \in \{1, 2, \dots, k\}$  of a group  $G_i$ , and the exponent  $s \in \mathbb{R}_{\geq 0}$  that parameterizes the Zipf distribution, the probability to sample an instance for group  $G_i$  can be expressed as

$$P_{k;s}(r_i) = \frac{1}{r_i^s H_{k,s}}, \tag{1}$$

where  $H_{k,s} = \sum_{j=1}^k \frac{1}{j^s}$  is the  $k$ -th harmonic number [Sc12]. Thereby, we can control the imbalance degree of the classes or groups using the parameter  $s$  of the distribution. As we use  $PD$  for two different purposes in our approach, we use the parameters  $s_C$  and  $s_G$  to describe the imbalance degrees for the classes and groups, respectively.

**Example:** Figure 2 shows an example of the Zipf distribution and the influence of the parameter  $s$ . We focus on the distribution of different groups, but it is analogous for the distribution of the classes. In this example, we assign the number of instances to  $k = 10$  groups. We show the probability to sample an instance for the  $i$ -th group on the y-axis. Figure 2a shows the probability to sample an instance for a group with the default setting  $s = 1$ . The probability to sample the first group is around 35% and for the second group

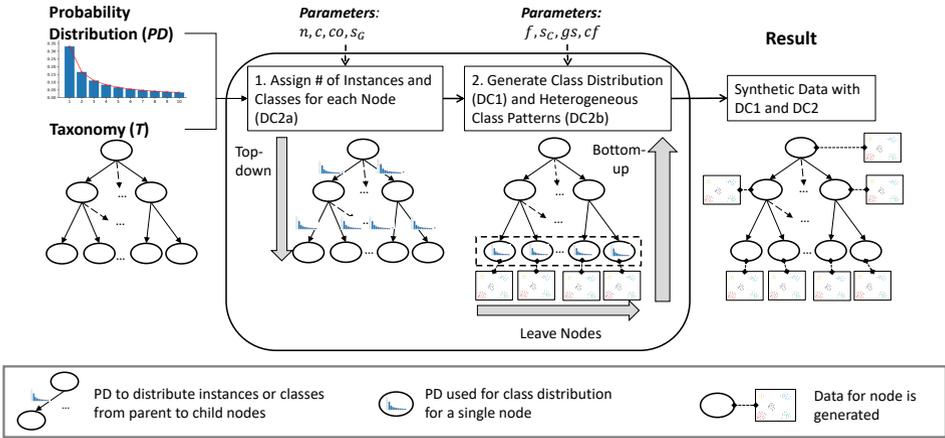


Fig. 3: General overview of the two main steps of our approach.

around 17.5%. Hence, if we sample 100 times on this distribution, we have one group that gets around 35 instances, a second group with around 18 instances, and so on. In particular, for  $s = 1$ , the element that occurs most often occurs twice as often as the second-most occurring element [Sc12]. Furthermore, we show the effect of varying the exponent  $s$ : Figure 2b shows a more imbalanced setting with  $s = 1.5$  and Figure 2c a more balanced setting with  $s = 0.5$ . In general, a setting with  $s < 1$  leads to more balanced and  $s > 1$  to more imbalanced distributions. A value  $s = 0$  describes an equal probability for each group.

### 4.3 Data Generation

Figure 3 gives a general overview of our two-step approach: First, we generate imbalanced groups (DC2a). To this end, we assign the number of instances and classes for each group, i. e., for each node in the taxonomy. Second, we generate an imbalanced class distribution for each node (DC1) and ensure the heterogeneity of the class patterns among the groups (DC2b). Note that due to DC2b, the heterogeneous class patterns are specific for each group. Thus, we first have to generate the groups and subsequently the classes and patterns within them so that we can directly ensure that the class patterns are different between the groups.

#### 4.3.1 Assign Number of Instances and Classes

In the first step, we aim to generate imbalanced groups (DC2a), i. e., we assign the number of instances and classes to all groups of the taxonomy. Algorithm 1 outlines the procedure for this first step. The inputs are the number of instances ( $n$ ), number of classes ( $c$ ), the class overlap across the groups ( $co$ ), the taxonomy ( $T$ ), the probability distribution ( $PD$ ), and the

parameter for the group imbalance degree ( $s_G$ ). First, we initialize the root node in line 1. Since the root node should comprise the entire dataset  $\mathcal{X}$ , we assign the overall numbers of instances  $n$  and classes  $c$  to the root node. Furthermore, we initialize the set of *nodes* that we traverse in the following with the root node (line 2). Hence, as long as we have nodes to traverse (line 3), we pick and remove the next *node* from the nodes set (line 4). If *node* does not have any child nodes, i. e.,  $n\_children = 0$ , we continue with the next iteration of the while loop (lines 6 - 8).

If the current *node* has children, we assign the number of instances and the number of classes for all its child nodes. To this end, we use the *PD* by calling the function `SAMPLE_PD(PD, node.n, n_children,  $s_G$ )` (line 9). We use this function to distribute the number of instances of a parent node (*node.n*) among its child nodes (lines 16 - 23). For each individual instance, we decide to which group it belongs using *PD*. Thus, we sample *node.n* times from the *PD* and sample each time one of the child nodes  $1, \dots, n\_children$ . We control the imbalance of sampling the groups with the parameter  $s_G$ . In the `SAMPLE_PD()` function, we first initialize a *counter* list that keeps track of the number of instances for each group (line 17). Subsequently, we sample one of the groups from *PD* and update the count for that group (lines 19 and 21). Finally, we return a list *n\_instances* that contains for each group the number of instances according to the *PD*. As this is an exponential distribution, we get an imbalanced distribution of the number of instances across all nodes for  $s_G > 0$ .

In line 10, we do the same procedure for the classes, i. e., we derive a list *n\_classes* that specifies the number of classes for each group. However, as mentioned in Section 2.2, classes typically occur in multiple groups. Thus, we add a factor of  $co \geq 1$  to control the number of classes among the groups. For  $co = 1$ , the classes are disjoint among the groups. For  $co > 1$ , classes may occur in multiple groups. In line 11, we set the number of instances and the number of classes as attributes of the child nodes. That is, we update the attribute of the child nodes with the corresponding samples of the lists *n\_instances* and *n\_classes* in the `UPDATE(children, n_instances, n_classes)` function (lines 24 - 30). In line 12, we add the child nodes to the *nodes* set to traverse them as well. Finally, we return the modified taxonomy, where each node contains the number of instances and the number of classes.

### 4.3.2 Assign Class Distribution and Generate Data

In the second step, we assign the class distributions and generate the data in a bottom-up manner. To this end, we use the information that we assigned in the previous step to each node. We first generate the data on all leaf nodes and subsequently pass the data upwards to the parent nodes.

Algorithm 2 outlines our procedure. First, we retrieve the leaf nodes from the taxonomy (line 1). Further, we initialize a key-value map to store the current feature limits (line 2). Then, we iterate over each leaf node (lines 3 - 17). We first retrieve the number of instances, number of classes, and the actual class labels from each leaf node (line 4). In line 5, we

---

**Algorithm 1** Algorithm to assign the number of instances and classes.

---

**Input:** T: Tree-structured taxonomy,*n*: Number of instances for the entire data,*c*: Number of classes for the entire data,*co*: Class overlap across groups,*PD*: Probability distribution,*s<sub>G</sub>*: Value for the group imbalance degree used by *PD*.**Output:** T: Tree-structured taxonomy that has the number of instances and classes assigned as attributes on each node.

```
    ▶ Initialize root node and nodes set
1: root.n ← n; root.c ← c; root.classes ← {1, ..., c};
2: nodes ← {root}
    ▶ Iterate while we have nodes
3: while nodes ≠ {} do
    ▶ Get and remove node from nodes set
4: node ← nodes.pop()
5: n_children ← |node.children|
6: if n_children == 0 then
7:     continue                                ▶ Leaf node, so continue with next node
8: end if
    ▶ Draw the number of instances for each child node
9: n_instances ← SAMPLE_PD(PD, node.n, n_children, sG)
    ▶ Draw the number of classes for each child node
10: n_classes ← SAMPLE_PD(PD, node.c * co, n_children, sG)
    ▶ Update n_instances and n_classes of child nodes
11: UPDATE(node.children, n_instances, n_classes)
12: nodes.APPEND(node.children)
13: end while
14: return T
15: procedure SAMPLE_PD(PD, k, n_groups, s)
16:   counter ← [0, ..., 0]                       ▶ Initialize counter of size n_groups
17:   for i = 1, ..., k do
18:     group ← PD(n_groups, s, i)             ▶ Sample group from PD
19:     counter[group] ← counter[group] + 1
20:   end for
21:   return counter
22: end procedure
23: procedure UPDATE(children, n_instances, n_classes)
24:   for i = 1, ..., |children| do
25:     child ← children[i]
26:     child.n ← n_instances[i]
27:     child.c ← n_classes[i]
28:   end for
29: end procedure
```

---

ensure the multi-class imbalance (DC1). Here, we assign the class occurrences, i. e., how often a particular class occurs in the dataset. To this end, we use  $PD$  to decide for each instance the associated class, similar as for the groups in Algorithm 1. Hence, we sample  $n$  times one of the classes  $1, \dots, c$  via the distribution  $PD$  and the function `SAMPLE_PD` (cf. lines 18 - 25 in Algorithm 1). In lines 6 and 7, we ensure the heterogeneity of the class patterns (DC2b). First, we pick  $cf$  characteristic features from the current group, i. e., the features that separate the current group from all other groups. To separate the groups, we ensure that the current group has different value ranges for the characteristic features than the other groups. To this end, we increment the feature limits with the  $gs$  parameter to control the differences in the value ranges between the groups.

In line 8, we generate the data using the input parameters  $n, c$ , the number of features  $f$ , and the list `class_occurrences`. That is, we generate the feature values for all instances with their associated class labels. To this end, we can use any multivariate probability distribution to generate the data that supports these inputs as parameters. As existing domain-agnostic data generators (cf. Section 3) already support generating data using different probability distributions, we can use them for that purpose. For example, the approach from Guyon [Gu03] may be used to generate the feature values and feature ranges for specific class labels according to a Gaussian distribution. However, this multivariate probability distribution can also be altered by the user to capture different feature correlations of real-world applications.

To guarantee the heterogeneity of class patterns, we ensure that each class has different value ranges in different groups for certain characteristic features. Such feature dependencies in form of characteristic features also appear frequently in industrial application scenarios that comprise heterogeneous groups [Wu16, HRM19, KRM19]. Therefore, we ensure in lines 9 and 10 that the groups have different value ranges according to the picked features for each group and the current `feature_limits`. To this end, we normalize the feature values of all instances, i. e., we normalize the values of each feature in  $\mathcal{X}$  into  $[0;1]$  (line 9). Subsequently, we add the current limits of the features (line 10). In line 11, we store  $\mathcal{X}$  in the current node. In lines 12 - 15, we update the data and the class labels of the parent nodes. That means we traverse the parent node and append the data  $\mathcal{X}$  of the child node to the currently stored data of the parent node. Finally, we return the taxonomy, where we set the data for each node.

## 5 Evaluation

In this section, we discuss the evaluation results for our approach, i. e., whether our approach is able to generate data synthetically that comply with the data characteristics DC1 and DC2. First, we describe the setup of our evaluation. Subsequently, we discuss to which extent our data generator can generate different manifestations of the data characteristics DC1, DC2a, and DC2b using different parameterizations.

**Algorithm 2** Algorithm to generate the data bottom-up.

**Input:**  $f$ : Number of features to generate,  
 $s_C$ : Imbalance degree for the class distribution,  
 $gs$ : Group separation,  
 $cf$ : Number of characteristic features to use for each group,  
 $PD$ : Probability distribution,  
 $T$ : Tree-structured taxonomy, where the numbers of instances and classes as well as the class occurrences are defined for each node with Algorithm 1.

**Output:**  $T$ : Tree-structured taxonomy, where we assigned for each leaf node how often each class occurs.

```

1:  $nodes \leftarrow GET\_LEAF\_NODES(T)$ 
2:  $feature\_limits \leftarrow INIT\_DICTIONARY(\{1, \dots, f\}, 0, 1)$ 
3: for  $node \in nodes$  do
4:    $n \leftarrow node.n; c \leftarrow node.c; classes \leftarrow node.classes;$ 
    $\triangleright$  Draw the number of instances for each class
5:    $class\_occurrences \leftarrow SAMPLE\_PD(PD, n, classes, s_C)$ 
    $\triangleright$  Pick characteristic features for this group
6:    $features \leftarrow PICK\_RANDOM\_FEATURES(\{1, \dots, f\}, cf)$ 
    $\triangleright$  Separate the groups
7:    $INCREMENT(feature\_limits, features, gs)$ 
    $\triangleright$  Actual data generation for each leaf node
8:    $X \leftarrow GENERATE\_DATA(n, c, f, class\_occurrences)$ 
9:    $X \leftarrow$  Normalize  $X$  into  $[0; 1]$ 
    $\triangleright$  Add for each feature its current limits
10:   $X \leftarrow X + feature\_limits$ 
    $\triangleright$  Store data in current node
11:   $node.X \leftarrow X;$ 
    $\triangleright$  add  $X$  to parent node
12:  while  $HAS\_PARENT(T, node)$  do
13:     $node \leftarrow GET\_PARENT(T, node)$ 
14:     $node.X \leftarrow APPEND(node.X, X)$ 
15:  end while
16: end for
17: return  $T$ 

```

## 5.1 Evaluation Setup

**Implementation.** Our prototypical implementation is available on Github<sup>7</sup>. For the evaluation, we use a taxonomy that we derived from a real-world use case regarding end-of-line testing of complex truck engines [HRM20]. A simplified excerpt of this taxonomy is shown in Figure 1. The taxonomy has three levels and 26 groups at the bottom level of the hierarchy. For more details on the taxonomy, we refer to our repository.

**Evaluation of the data characteristics.** The goal of our evaluation is to show that the generated data of our approach comply with the data characteristics DC1, DC2a, and DC2b. We evaluate the presence of each data characteristic with different evaluation measures, i. e., class imbalance measures for DC1, group imbalance measures for DC2a, and complexity

<sup>7</sup> Prototypical implementation: <https://github.com/IPVS-AS/DataGenerator>

Tab. 2: Overview of parameters that we discuss regarding their influence on the data characteristics. The bold values indicate the parameters that we vary for the respective characteristics, while we use default values for the other parameters.

| Data Characteristics                   | Parameter values                      |  |   |   |
|--|---------------------------------------|--|---|---|
|  | Class imbalance ( $s_C$ )             | Group imbalance ( $s_G$ )              | #charact. features ( $cf$ )                   | Group separation ( $gs$ )                             |
| Multi-Class Imbalance (DC1)            | <b>{0, 1, 2<sup>†</sup>, 3, 4, 5}</b> | 1                                      | 10  | 0.25  |
| Group Imbalance (DC2a)                 | 2                                     | <b>{0, 0.5, 1, 1.5<sup>†</sup>, 2}</b> | 10  | 0.25  |
| Heterogeneity of Class Patterns (DC2b) | 2                                     | 1                                      | <b>{1, 5, 10<sup>†</sup>, 15, 20, 25, 30}</b> | <b>{0<sup>†</sup>, 0.05, 0.1, 0.25, 0.5, 0.75, 1}</b> |

<sup>†</sup> Parameter values that lead to similar statistics as the real-world data in the work of Hirsch et al. [HRM19, HRM20].

Tab. 3: Gini coefficient values for the generated datasets with varying  $s_C$  parameter.

| Class Imbalance ( $s_C$ ) | $s_C = 0$ | $s_C = 1$ | $s_C = 2$ | $s_C = 3$ | $s_C = 4$ | $s_C = 5$ |
|---------------------------|-----------|-----------|-----------|-----------|-----------|-----------|
| Gini Coefficient          | 28%       | 41%       | 60%       | 69%       | 72%       | 73%       |

measures for DC2b. To this end, we also vary the parameter values of our data generator to show that it is capable of generating different manifestations of the data characteristics. This is an essential requirement for generating data that may serve as basis for benchmarks of classification algorithms. Furthermore, we also compare the statistics of the generated data with the statistics of a real-world data set of Hirsch et al. [HRM19, HRM20].

**Parameters.** Our data generator has eight parameters in total. As our goal is to evaluate which manifestations of the data characteristics it can generate, we only vary the parameters that have a strong influence on these characteristics. Therefore, we generate data with fixed parameters  $n = 1000$  instances,  $f = 40$  features,  $c = 30$  classes, and  $co = 1.5$ . Table 2 shows the parameters that we vary to study the influence on individual data characteristics. To evaluate the manifestation of a single data characteristic, we only vary and discuss the results for the parameters that influence this particular characteristic, using default values for the other parameters (cf. Table 2).

## 5.2 Evaluation of Multi-Class Imbalance (DC1)

We evaluate the presence of the data characteristic DC1 w.r.t. the class imbalance and the accuracy for minority and majority classes of a classification model.

**Class imbalance.** There is no consensus on proper statistical metrics to determine the degree of class imbalance within data [Fe13]. Yet, an often-used metric for inequality that takes the value of 100% in case of total imbalance and 0% for total balance is the Gini coefficient [Co00]. Thus, we use the Gini coefficient to measure the imbalance of the classes for the generated datasets.

Table 3 shows the Gini coefficients for the generated datasets with different  $s_C$  parameter values. We observe that the Gini coefficient values for the generated data vary from 28% for

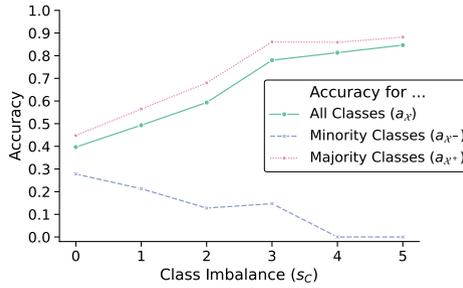


Fig. 4: Overview of accuracy for all classes ( $a_X$ ), and separately for minority ( $a_{X^-}$ ) and majority classes ( $a_{X^+}$ ) for the different  $s_C$  values.

$s_C = 0$  to 73% for  $s_C = 5$ . Thus, the class imbalance is increasing with an increasing value of the  $s_C$  parameter. We also observe that the Gini coefficient is increasing faster for lower  $s_C$  values, i. e., it is increasing by 19%-points from  $s_C = 1$  to  $s_C = 2$ , but only by 2%-points from  $s_C = 4$  to  $s_C = 5$ . The reason why the coefficient does not increase as much for higher  $s_C$  values is due to the calculation of the Gini coefficient. For example, to achieve a Gini coefficient of 100%, the generated data would contain instances for solely one class [Co00]. So, for the generated data to have a higher Gini coefficient, some of the classes must not appear in the data at all. However, in our implementation, we ensure that each class occurs at least once. The real-world data in the work of Hirsch et al. [HRM19, HRM20] shows a Gini coefficient of 55%. Thus, with  $s_C = 2$ , we can generate data with a similar class imbalance as real-world data that also comprise DC1 and DC2.

**Accuracy of minority and majority classes.** As described in Section 2.1, the accuracy of a classifier typically correlates with the accuracy for the majority classes in multi-class imbalance problems. In particular, the accuracy for minority classes usually decreases as the degree of class imbalance increases [HG09, WY12]. Therefore, we examine whether the generated data by our generator also exhibit this trend. To measure the accuracy, we split each generated dataset into 70% training data  $\mathcal{X}_{train}$  and 30% test data  $\mathcal{X}_{test}$ . Thereby, we preserve the same class distribution in both data subsets. Subsequently, we train a classifier  $M_X$  on  $\mathcal{X}_{train}$  and denote the accuracy of  $M_X$  on  $\mathcal{X}_{test}$  as  $a_X$ . We also denote with  $a_{X^+}$  the accuracy among only the instances of the majority classes and with  $a_{X^-}$  for the minority classes. We declare a class as minority class if it has less instances than the median number of instances of all classes and otherwise as majority class. We use Random Forest as classification model due to its robustness regarding the characteristics [HRM19, HRM20].

Figure 4 shows the accuracy of the minority and the majority classes for each generated dataset. We observe that the more imbalanced the data, the higher is the accuracy for the whole dataset ( $a_X$ ) and for the majority classes ( $a_{X^+}$ ). The reason for this high correlation between  $a_X$  and  $a_{X^+}$  is that the majority classes have by far the biggest share of instances of the data. So, they also contribute much more to the overall accuracy than the minority

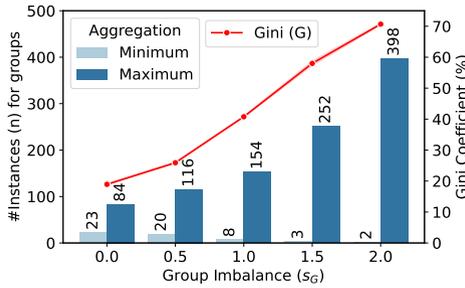


Fig. 5: Overview of the number of instances per group as well as the Gini coefficient values for different  $s_G$  values.

classes. However, the accuracy of the minority classes decreases with an increasing class imbalance. As mentioned above, this is the expected behavior for the accuracy in multi-class imbalance problems. Hence, when we control the imbalance with the  $s_C$  parameter, the generated data has the expected accuracy curves regarding minority and majority classes. Thus, we are able to generate various and proper manifestations of DC1.

### 5.3 Evaluation of Group Imbalance (DC2a)

To evaluate the presence of DC2a, we examine the imbalance of the generated groups and measure the extent of representation bias in data, i. e., if certain groups are underrepresented. To this end, we vary the  $s_G$  parameter as it has the highest influence on DC2a. We use the Gini coefficient to measure the degree of imbalance for the generated groups, i. e., we apply the Gini coefficient to the group labels. Further, we report aggregated statistics about the number of instances for all groups. Figure 5 shows the minimum and the maximum number of instances of all groups as well as the Gini coefficient for the groups.

We observe an increase in the Gini coefficient for increasing  $s_G$  values. For  $s_G = 0$ , we have a Gini coefficient of around 20%, while it is around 70% for  $s_G = 2$ . Thus, we are able to control the imbalance of the groups (DC2a). The maximum number of instances for each group is also increasing for higher  $s_G$  values. In particular, for  $s_G = 0$ , we have a maximum of 84 instances for a group, while it is 398 for  $s_G = 2$ . On the other side, the minimum number of instances for a group is decreasing from 21 to 2. For the lowest parameter value  $s_G = 0$ , the Zipf distribution assigns the same probability to all groups. However, not exactly the same number of instances are assigned to all groups because it is still a random distribution. Therefore, slight deviations occur. In our approach, small deviations can occur at each level of the taxonomy due to our top-down procedure (cf. Algorithm 1), resulting in a Gini coefficient of 20% for  $s_G = 0$ . For the highest parameter value  $s_G = 2$ , some groups are underrepresented as they solely occur 2 times in the generated dataset. This shows that we can generate and control different manifestations of a representation bias in the data

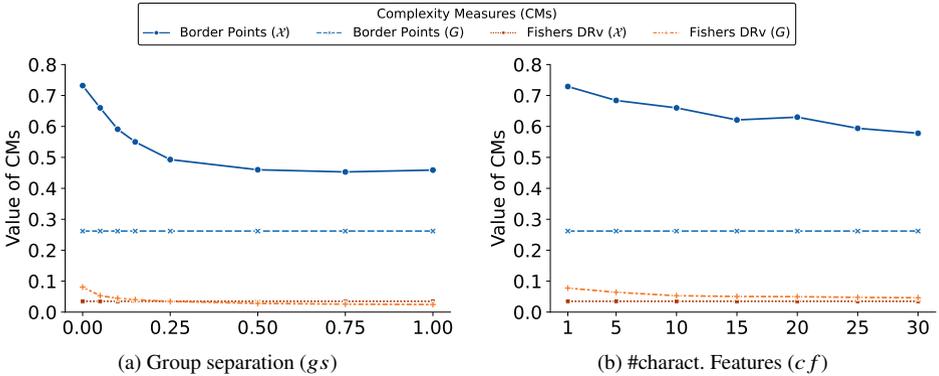


Fig. 6: Values of complexity measures for the entire data  $\mathcal{X}$  and averaged over the groups  $G$  regarding (a)  $gs$  and (b)  $cf$ .

with the  $s_G$  parameter (DC2a). The real-world data from the use case of end-of-line testing have a Gini coefficient of 54% regarding the group distribution, while the rarest group has 2 instances and the most frequent group has 300 instances [HRM19, HRM20]. Thus, we are able to generate a similar group distribution with  $s_G = 1.5$ .

#### 5.4 Evaluation of Heterogeneous Class Patterns (DC2b)

To measure the manifestation of the characteristic DC2b, we measure the effects of the aggregation bias in data (cf. Section 2). So, we focus on the difference in carrying out a data analysis (1) for the whole data and (2) for each group separately. To this end, we examine (i) the complexity of the classification problem in the generated data and (ii) the accuracy of a classification model on the generated data.

**Difference of complexity measures.** We use commonly used complexity measures [HB02] to assess the complexity of the classification problem independently of a specific classification algorithm. For sake of clarity, we focus in our discussion on the results of two commonly used complexity measures that are accompanying symptoms of the characteristics DC1 and DC2: i) The Directional-Vector Maximum Fishers Discriminant Ratio (*Fishers DRv*) measures how well the classes can be separated by the feature values. ii) The fraction of *Border Points* measures the fraction of all instances where the nearest instance belongs to a different class. We note that for both complexity measures, lower values indicate simpler classification problems, i. e., the classes are better separable or the data has less border points. We compare the complexity measures on the entire data  $\mathcal{X}$  and the average values over all groups (DC2). More formally, for each of the complexity metrics  $CM$ , we denote the value of the  $CM$  on  $\mathcal{X}$  as  $CM(\mathcal{X})$ . We calculate the average over the groups  $G = \{G_1, \dots, G_k\}$  as  $CM(G) = \frac{1}{k} \sum_{i=1}^k CM(G_i)$ .

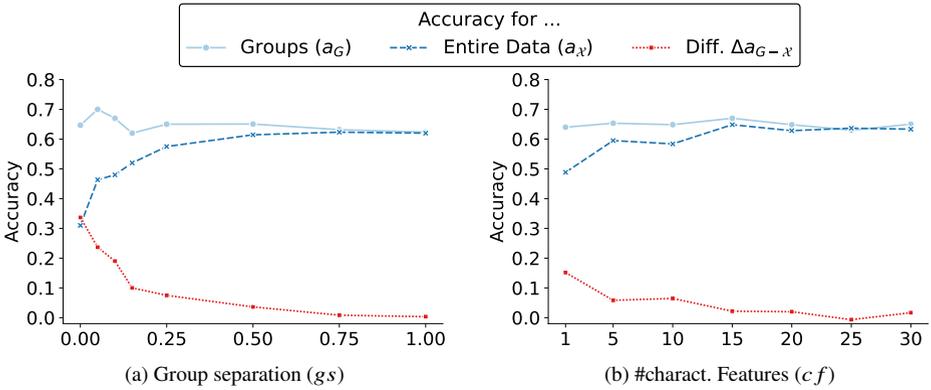


Fig. 7: Overview of accuracy results for the varying parameter configurations for our data generator. The accuracy is shown for training the model on the whole data or solely on the groups, as well as the difference in the accuracy of both.

Figure 6 shows the difference of the complexity measures averaged over all groups and on the entire data. We observe in Figure 6a that for  $g_s \leq 1$ , the whole data  $\mathcal{X}$  comprise more border points compared to the average over all groups  $G$ . In particular, the highest absolute difference is around 25%-points for  $g_s = 0$ , while it is decreasing with higher  $g_s$  values. Thus, different classes are more often close to each other on the whole data compared to the data subsets of all groups for  $g_s = 0$ . Nevertheless, this difference decreases for higher  $g_s$  values. The reason is that for  $g_s = 0$ , the groups are not separated at all. As a result, classes from different groups have the same feature ranges and are therefore more likely to border on each other. For  $g_s > 0$ , the groups have different value ranges for their characteristic features and thus a slight increase of  $g_s$  leads to a high decrease in the border points. Yet, for higher  $g_s$  values, the border points for  $\mathcal{X}$  converge to around 45%.

We observe a similar trend for Fishers DRv, i. e., for  $g_s = 0$ , the average value over the groups is less compared to the entire data  $\mathcal{X}$ . Thus, the classes are more easily separable, when considering the individual data subsets of the groups. Yet, for higher  $g_s$  values, Fishers DRv is decreasing on  $\mathcal{X}$  and is even less than the average over the groups for  $g_s \geq 0.5$ . Hence, the classes are better separable on  $\mathcal{X}$  for  $g_s \geq 0.5$ . This concludes that there is a similar correlation for the Fishers DRv as for the border points regarding the  $g_s$  value. For the  $cf$  parameter (cf. Figure 6b), we observe similar trends as for  $g_s$ . Thus, we do not discuss these results in more detail.

**Difference in accuracy.** We also measure the difference in accuracy averaged over all groups and on the entire data. To this end, similar as in Section 5.2, we train a Random Forest classifier  $M_{\mathcal{X}}$  on  $\mathcal{X}_{train}$  and report the accuracy on the test set  $\mathcal{X}_{test}$  as  $a_{\mathcal{X}}$ . Further, we train a set of classifiers  $\mathcal{M}_G = \{M_1, \dots, M_g\}$ , where each  $M_i$  is a Random Forest classifier trained separately on a group  $G_i \subset \mathcal{X}_{train}$ . In this case, we predict the class label for each

test instance  $x_{test} \in \mathcal{X}_{test}$  that belongs to group  $G_j$  with the model  $M_j$ . Thus, we denote the accuracy of  $\mathcal{M}_G$  for  $\mathcal{X}_{test}$  with  $a_G$ . Further, we define the difference in accuracy as  $\Delta a_{G-\mathcal{X}} = a_G - a_{\mathcal{X}}$ .

Figure 7 shows the accuracy results. For the  $gs$  parameter (cf. Figure 7a), we observe that the highest difference in accuracy is obtained for  $gs = 0$ , which is more than 30%-points. Thus, we have the strongest effect of an aggregation bias for  $gs = 0$ . Yet, the difference in the accuracy is decreasing for higher  $gs$  values, i. e., for  $gs = 1$  it is about 5%-points. The reason can be seen in the previous results for the complexity measures: The data has less border points and the classes are more separable on  $\mathcal{X}$  for higher  $gs$  values. Thus, it is easier for a classification model to predict the classes more accurately on  $\mathcal{X}$ . Again, we observe similar results for the  $cf$  parameter in Figure 7b. We note that Random Forest achieves an accuracy of 33% on real-world data that comprise both characteristics [HRM19, HRM20]. Thus, we can generate data with similar accuracy results using  $gs = 0$ .

Concluding, the results show that we are able to control the difference of the complexity measures and the accuracy with the  $gs$  and the  $cf$  parameters. In other words, our data generator is able to control the heterogeneity of the class patterns, i. e., the aggregation bias in the generated data (DC2b).

## 6 Conclusion

The contribution of this paper is an approach to generate synthetic data comprising two data characteristics that often occur in real-world use cases: multi-class imbalance (DC1) and heterogeneous groups (DC2). The actual manifestations of these data characteristics are domain-specific, i. e., dependent on the actual real-world use case. Therefore, our approach uses a taxonomy model and a two-step process to generate data that reflect the characteristics of a given real-world use case. A taxonomy is the simplest form of knowledge model to organize real-world entities in domain-specific groups and it can be found in various domains. So, our approach is not limited to a specific domain. In our evaluation, we unveil that the generated data comprises the characteristics DC1 and DC2 together. Moreover, the parameters of our data generator may be steered to reflect different manifestations of these characteristics. Our approach builds the fundamental basis for future work to create a benchmark that evaluates machine learning and data engineering approaches systematically on data with the characteristics DC1 and DC2. Thereby, correlations between different manifestations of the characteristics and the performance of approaches can be examined.

**Acknowledgements.** This research was performed in the project 'VALID-PARTITION' as part of the Software Campus program, which is funded by the German Federal Ministry of Education and Research (BMBF) under Grant No.: 01IS17051. This work was also supported by the Ministry of Science, Research, and the Arts of the State of Baden-Württemberg within the sustainability support of the projects of the Excellence Initiative II.

## Bibliography

- [AK04] Agard, Bruno; Kusiak, A.: Data-Mining-based Methodology for the Design of Product Families. *International Journal of Production Research*, 42(15):2955–2969, 2004.
- [AI11] Alcalá-Fdez, Jesús et al.: KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *Multiple-Valued Logic and Soft Computing*, 2011.
- [Ch20] Chan, Stephanie et al.: Machine Learning in Dermatology: Current Applications, Opportunities, and Limitations. *Dermatology and Therapy*, 2020.
- [Ci09] Cimiano, Philipp et al.: Ontology learning. In: *Handbook on ontologies*, pp. 245–267. Springer, 2009.
- [Co00] Cowell, Frank: *Measuring Inequality*. 3<sup>th</sup> edition, 2000.
- [DC19] Dahmen, Jessamyn; Cook, Diane: SynSys: A synthetic data generation system for healthcare applications. *Sensors*, 19(5):1181, 2019.
- [DG17] Dua, Dheeru; Graff, Casey: , UCI Machine Learning Repository, 2017.
- [Fe13] Fernández, Alberto et al.: Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-Based Systems*, 42:97–110, apr 2013.
- [Fe20] Fernandes, Ederson Carvalhar et al.: Flexible Production Data Generator for Manufacturing Companies. *Procedia Manufacturing*, 2020.
- [Fr11] Frasch, Janick V. et al.: A Bayes-true data generator for evaluation of supervised and unsupervised learning methods. *Pattern Recognition Letters*, 32(11):1523–1531, 2011.
- [FS18] Fränti, Pasi; Sieranoja, Sami: K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48:4743–4759, 2018.
- [Ga12] Galar, Mikel et al.: A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2012.
- [GBC16] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron: *Deep learning*. MIT press, 2016.
- [Gr22] Gröger, Christoph: Industrial Analytics — An Overview. *it – Information Technology*, 64(1–2):55–65, 2022.
- [Gu03] Guyon, Isabelle: Design of experiments for the NIPS 2003 variable selection benchmark. 2003.
- [HB02] Ho, Tin Kam; Basu, Mitra: Complexity measures of supervised classification problems. *IEEE TPAMI*, 2002.
- [HG09] He, Haibo; Garcia, E.A.: Learning from Imbalanced Data. *IEEE TKDE*, 21(9):1263–1284, sep 2009.
- [Ho21] Hogan, Aidan et al.: Knowledge graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, 12(2):1–257, 2021.

- [HRM19] Hirsch, Vitali; Reimann, Peter; Mitschang, Bernhard: Data-driven fault diagnosis in end-of-line testing of complex products. In: IEEE DSAA. 2019.
- [HRM20] Hirsch, Vitali; Reimann, Peter; Mitschang, Bernhard: Exploiting domain knowledge to address multi-class imbalance and a heterogeneous feature space in classification tasks for manufacturing data. VLDB, 2020.
- [Ig19] Iglesias, Félix et al.: MDCGen: Multidimensional Dataset Generator for Clustering. Journal of Classification, 2019.
- [Ja04] Jablonski, Nina: The Evolution of Human Skin and Skin Color. Ann. Review of Anthropology, 33:585–623, 2004.
- [KBT11] Köksal, Gülsel; Batmaz, İnci; Testik, Murat Caner: A review of data mining applications for quality improvement in manufacturing industry. Expert Systems with Applications, 38(10):13448–13467, sep 2011.
- [KM16] Kassner, Laura; Mitschang, B.: Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics. In: EDBT. 2016.
- [KRM19] Kiefer, Cornelia; Reimann, Peter; Mitschang, Bernhard: A Hybrid Information Extraction Approach Exploiting Structured Data Within a Text Mining Process. In: BTW 2019. Gesellschaft für Informatik, Bonn, pp. 149–168, 2019.
- [KU15] Khan, Aunsia; Usman, Muhammad: Early Diagnosis of Alzheimer’s Disease Using Machine Learning Techniques: A Review Paper. In: Proc. of the 7<sup>th</sup> International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K). Lisbon, Portugal, pp. 380–387, 2015.
- [LKJ02] Lundin, Emilie; Kvarnström, Håkan; Jonsson, Erland: A Synthetic Fraud Data Generation Methodology. In: Information and Communications Security. 2002.
- [Me21] Mehrabi, Ninareh et al.: A Survey on Bias and Fairness in Machine Learning. ACM Comput. Surv., 54(6), jul 2021.
- [MGTM20] Maitín, Ana María; García-Tejedor, Alvaro José; Muñoz, Juan Pablo Romero: Machine Learning Approaches for Detecting Parkinson’s Disease from EEG Analysis: A Systematic Review. Applied Sciences, 10(23), 2020.
- [PWV16] Patki, Neha; Wedge, Roy; Veeramachaneni, Kalyan: The Synthetic Data Vault. In: IEEE DSAA. 2016.
- [RHW21] Roh, Yuji; Heo, Geon; Whang, Steven Euijong: A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. IEEE TKDE, 33(4):1328–1347, apr 2021.
- [SB21] Steinbuss, Georg; Böhm, Klemens: Benchmarking Unsupervised Outlier Detection with Realistic Synthetic Data. ACM TKDD, 2021.
- [Sc12] Schervish, Mark J: Theory of statistics. Springer Science & Business Media, 2012.
- [SG21] Suresh, Harini; Guttag, John: A Framework for Understanding Sources of Harm throughout the Machine Learning Life Cycle. In: Equity and Access in Algorithms, Mechanisms, and Optimization. EAAMO ’21, 2021.

- [SGG18] Suresh, Harini; Gong, Jen J.; Guttag, John V.: Learning Tasks for Multitask Learning: Heterogenous Patient Populations in the ICU. In: SIGKDD. 2018.
- [SH05] Steinley, Douglas; Henson, Robert: OCLUS: An Analytic Method for Generating Clusters with Known Overlap. *Journal of Classification*, 2005.
- [So91] Sowa, John F.: Principles of Semantic Networks. Explorations in the Representation of Knowledge. Morgan Kaufmann, 1991.
- [Su14] Sun, Chong et al.: Chimera: Large-Scale Classification using Machine Learning, Rules, and Crowdsourcing. VLDB, 2014.
- [SWK09] Sun, Yanmin; Wong, Andrew; Kamel, Mohamed: Classification of Imbalanced Data: A Review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4):687–719, 2009.
- [ULP19] Ustun, Berk; Liu, Yang; Parkes, David: Fairness without harm: Decoupled classifiers with preference guarantees. In: ICML. 2019.
- [Va14] Vanschoren, Joaquin et al.: OpenML: Networked Science in Machine Learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, jun 2014.
- [Wi20] Wilhelm, Yannick et al.: Data Science Approaches to Quality Control in Manufacturing: A Review of Problems, Challenges and Architecture. In: Proc. of the 14<sup>th</sup> Symposium on Service-Oriented Computing (SummerSOC). *Communications in Computer and Information Science (CCIS)*. Springer-Verlag, pp. 45–65, 2020.
- [WLL21] Wang, Jialu; Liu, Yang; Levy, Caleb: Fair Classification with Group-Dependent Label Noise. In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. FAccT '21, Association for Computing Machinery, New York, NY, USA, p. 526–536, 2021.
- [Wu16] Wuest, Thorsten et al.: Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, jan 2016.
- [WY12] Wang, Shuo; Yao, Xin: Multiclass Imbalance Problems: Analysis and Potential Solutions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(4):1119–1130, 2012.



# No Mayfly: Detection and Analysis of Long-term Twitter Trends

John Ziegler<sup>1</sup> Michael Gertz<sup>1</sup>

**Abstract:** The focus of social media is characterized by stories about short-lived breaking news. Often, such “mayflies” make it hard to keep track of more profound topics that are prevalent over a long period of time. To provide such capabilities, we present a method to detect long-term trends based on temporal networks and community evolution. Connecting those methods with trend analysis approaches allows to study the temporal development of trends, their contextual information and how they are interrelated over time, which is of great benefit compared to existing work. Results obtained from a Twitter case study are discussed in detail and evaluated based on real-world event linkage, which proves the good functionality of the proposed method.

**Keywords:** Social Media Analytics; Temporal Networks; Trend Analysis; Twitter Data

## 1 Introduction

In today’s social media landscape discussed topics and attention are rapidly changing. It is hard to not get distracted by short-lived trends (“mayflies”) and instead keep focused on more profound and steady topics. In this work, inspired by the slow journalism movement [Le15], we do not analyze breaking news and trends of short attention but instead, focus on long-term trends. For this, a framework to detect and analyze long-term social media trends is outlined. It builds on existing work that is adopted and extended to fit the use case requirements. These extensions include: 1. Leveraging a temporal network model to study long-term trends, 2. Pruning of less prevalent nodes based on a power law degree distribution model, 3. Temporal tracking of hashtag communities via a core of central nodes, and 4. Appropriate visualizations to analyze the temporal development of found trends. The proposed methodology is applied to the German political Twitter-sphere to analyze long-term political trends. Thereby, the network-based approach allows to intuitively represent detected trends within their semantic context. In contrast to related work, e.g., the work by Chae and Park [CP18], our analysis specifically investigates semantic shifts of detected trends over time.

Regarding the used terminology, we do not refer to “trends” as they are often used in a time series analysis setting, e.g., [CC08, pp. 27-54]. Instead, trends in our social media analysis setting do come with a semantic meaning. Asur et al. describe trends as topics that

---

<sup>1</sup> Heidelberg University, Institute of Computer Science, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany; ziegler@informatik.uni-heidelberg.de, gertz@informatik.uni-heidelberg.de

“[...] capture the attention of a large audience [...]” [As11]. We follow this definition and start by taking Twitter hashtags as representatives of topics, which is in line with previous work, e.g., [As11] [BAE11]. According to Bhulai et al. [Bh12], these hashtags might also be clustered. As a result, we extend the previous definition of a “topic” and do not refer to it as a single hashtag, but as a community of hashtags. Tracking those communities of hashtags over time results in “temporal topics”. A topic can be said to make up a “trend” if its popularity is large enough (cf. [As11]) and is further called a “long-term” trend if it is prevalent over a sufficiently long time period. Together, we denote them as “long-term topical trends”. Further, for differentiation between short- and long-term trends, we refer to the concept of “news cycles” or rather “political information cycles” as described by Chadwick [Ch11]. These cycles describe news production processes and typically cover a time span of a few days. Topics that are discussed in the context of such short-lived media attention cycles are defined as short-term trends. In contrast, long-term trends describe topics that are prevalent in media for several weeks, months, or even years. This distinction is in line with past work, e.g., [Ha16].

This paper is structured as follows: First, in Section 2 related work is described and compared. Section 3 then covers the methodology concerning the detection of long-term trends. The proposed method is applied to a collected political Twitter dataset, and the according analysis is described and evaluated in Section 4. Finally, Section 5 gives a summary of the present framework and describes future work. Also, the source code used for the analysis steps is publicly available at the following URL: <https://github.com/jomazi/twitter-long-term-trends>.

## 2 Background

Most studies related to social media trend analysis focus on short-lived and mostly event-driven scenarios, e.g., [As11] and [BAE11]. Nevertheless, Chae and Park [CP18], as an example, apply topic detection to a long-term Twitter dataset and investigate trends within the corporate social responsibility domain. They study how the popularity of topics changes over time and how topics are interrelated. In contrast to their work, we focus on the political domain, specifically aiming to analyze topical shifts over time and follow a temporal network-based approach. Also related to trend analysis, Annamoradnejad and Habibi [AH19] study the trends published by Twitter itself. Thereby, they analyze the trending time as well as the trend’s re-occurrence over time. Further, Majdabadi et al. [Ma20] propose a graph-based Twitter trend extraction method and do not only take hashtags but also terms into account. Still, they do not track those trends over long time periods. Similarly, the work by Khan et al. [Kh21] is dealing with the detection as well as ranking of trends based on Twitter data. Some existing work from the field of information retrieval also approaches trend-related use cases. As an example, Hashvati et al. [Ha16] propose an online method to detect trends in a user search context. Notably, they also use social network communities as trend candidates and distinguish between short- and long-term

trends. Further, focused on classifying trends on Twitter, the work by Zubiaga et al. [Zu15] outlines a classification system of Twitter trends, along with methods to correctly identify a trend’s category at its initial stage. For trends, they rely on the official trends shown on the Twitter platform. These trends are short-living [Tw] and, are either related to news, ongoing events, memes or commemoratives [Zu15].

### 3 Methodology

In the following section, the methodology underlying the detection of long-term trends is outlined. For this, we first introduce the leveraged dataset in Section 3.1, then continue by describing the temporal network-based model formalism in Section 3.2 and outline the processing of the used hashtag co-occurrence networks in Section 3.3. Finally, Sections 3.4 and 3.5 cover the detection of topics and their tracking over time, which also leads to the extraction of topical long-term trends.

#### 3.1 Dataset

The EPINetz Twitter Politicians Dataset 2021 provides “[. . .] Twitter accounts of German parliamentarians, ministers, state secretaries, parties, and ministries on a state, federal, and European Union level for the year 2021” [Kö22]. We rely on the Twitter search API v2<sup>2</sup> to gather the raw tweets based on those user accounts. We collect tweets posted by the 2,449 accounts for the time range from January 2021 until July 2022 without filtering. In total, the dataset contains about 1.8 million tweets. Hashtags used in the tweets are taken as representatives of topics, which corresponds to the procedure of other works [As11] [BAE11]. We extract timestamped information about the (co-)occurrence of the hashtags from the unprocessed tweets and use them as the basis for detecting long-term topical trends.

#### 3.2 Temporal networks

Taking the timestamped information about hashtag (co-)occurrences as described above, temporal networks are created as aggregations based on a given time window. To formally describe the temporal snapshot networks we rely on the framework of multi-slice networks as outlined by Bianconi [Bi18, pp. 106-110]. A multi-slice temporal network is a special kind of multilayer network with each layer/slice representing a temporal snapshot of the complete network. As in our case, no interactions across snapshots exist, we focus on the intralink networks only, i.e., multi-slice networks without interlinks. Such a multilayer network  $M$  is defined as a tuple,  $M = (L, \mathcal{G})$ . It consists of the network layers  $L$  with  $|L| = l$ .

<sup>2</sup> Twitter Developer Platform: <https://developer.twitter.com/en/docs/twitter-api/tweets/search/introduction>; Accessed 28-12-22

A single layer is referred to as  $\ell \in L$ . Additionally,  $\mathcal{G}$  describes the time-ordered list of networks that are made up of the interactions within each of those layers:

$$\mathcal{G} = (G_1, G_2, \dots, G_\ell, \dots, G_l) \quad \text{with} \quad G_\ell = (V_\ell, E_\ell) \quad (1)$$

Each network  $G_\ell$  consists of a set of nodes  $V_\ell$ , which are in our case hashtags and their co-occurrences as edges  $E_\ell$ . Given that a multi-slice network  $M$  covers the interactions within a time period  $T$  and the time-window  $\Delta t$  is chosen as snapshot size, e.g., one month, there are  $l = T/\Delta t$  layers. Thereby, layer  $\ell$  captures the interactions that occur in the timeframe  $[(\ell - 1)\Delta t, \ell\Delta t)$ . Within such a layer  $\ell$  the degree of a node  $i$  is denoted as  $k_i^\ell$ . Further, for the *aggregated* network  $\tilde{G}$  of the multi-slice network, the temporal nature of the interactions is simply neglected and edges from all snapshots are taken into account.

### 3.3 Network processing

In contrast to mostly event- or breaking news-related short-term trends [Zu15], which are often represented by a single hashtag, long-term trends deal with more complex topics and can therefore be seen as communities of interrelated hashtags (see Section 1). To obtain more meaningful community networks and to further save computational costs during the community detection step (see Section 3.4), we focus on popular and highly connected hashtags. For this, less connected hashtags, i.e., with a low co-occurrence degree, are removed from the temporal networks. We take the median node degree per snapshot as a reference and remove all hashtags with a degree below this threshold from the according temporal network. An investigation of the degree distribution reveals its power law nature ( $k \propto k^{-\alpha}$ ). Therefore, we leverage the median as defined by Newman [Ne05]:

$$k_{med} = 2^{1/\alpha-1} k_{min} \quad (2)$$

An exemplary degree distribution is shown in Figure 1. The fitting procedure, for which the “powerlaw” package provided by Alstott et al. [ABP14] is used, reveals a power law exponent of 1.42 and according to that a median  $k_{med}$  of 5.31. In addition to the pruning step, the temporal snapshot networks are weighted. Ideally, respective edge weights reflect the semantic expressiveness of a hashtag and the strength of interrelations between hashtags. For this, we refer to Pointwise Mutual Information (PMI) [RN11]. Given that  $f_i^\ell$  describes the frequency of occurrence of node  $i$  during the timeframe covered by layer  $\ell$  and  $f_{ij}^\ell$  the frequency of co-occurrence of nodes  $i$  and  $j$ , the according PMI value is defined as:

$$\text{PMI}_{ij}^\ell = \ln \frac{f_{ij}^\ell}{f_i^\ell \cdot f_j^\ell} = w_{ij}^\ell \quad (3)$$

As indicated in Equation 3, those PMI values are used as co-occurrence edge weights  $w_{ij}^\ell$  between hashtag  $i$  and  $j$  in layer  $\ell$  of the temporal multi-slice network.

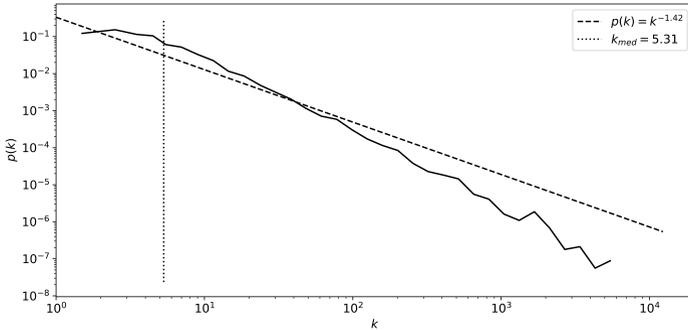


Fig. 1: Degree distribution of the January 2021 network snapshot

### 3.4 Detection of hashtag communities

Hashtags, i.e., the nodes of the temporal networks, are taken as representatives of topics [As11] [BAE11]. Further, according to Bhulai et al. [Bh12] in a comprehensive trend analysis framework related topics should be clustered. Therefore, we rely on methods developed in the field of community detection to find groups of densely interrelated hashtags. Those groups of hashtags then form a topic with all of its aspects as multiple hashtags might describe different semantic dimensions of the topic. To be precise, we leverage the Leiden community detection algorithm by Traag et al. [TWV19] and use the implementation as provided by the igraph software package [CN+06]. The community detection is applied to all layers of the temporal network described in Section 3.2.

### 3.5 Long-term trend detection

Of course, temporal communities of hashtags, i.e., temporal topics, as described in Section 3.4 do not yet make up a long-term topical trend. Asur et al. describe trends as topics that “[. . .] capture the attention of a large audience [. . .]” [As11], which means that trends need to reach a certain level of popularity. For this to measure, we take the accumulated count of hashtag occurrences per community and time window as trend scores. A community  $i$  in the network layer/slice  $\ell$  is given as a subset of hashtag nodes:  $C_i^\ell \subseteq V_\ell$ . Together with a mapping of those nodes to their respective occurrence counts for the given layer  $\ell$ ,  $o_\ell : V_\ell \rightarrow \mathbb{N}$ , we define the trend scores  $\tau$  as follows:

$$\tau(C_i^\ell) = \sum_{v \in C_i^\ell} o_\ell(v) \quad (4)$$

Those scores allow to rank detected trends by their popularity and, as an example, only the top- $n$  trends can be investigated. *Long-term* trends, opposed to short-lived trends, need to be prevalent over a sufficiently large time span. Therefore, detected hashtag communities need to be tracked over time. In their work, Lorenz et al. [Lo17] specifically propose a method to capture the dynamics of weighted hashtag co-occurrence networks. Not only does their method allow to track communities of hashtags across subsequent time steps, but also across further distant snapshots. Considering higher-order memory, i.e., taking the networks of multiple previous snapshots into account, their approach allows to overcome issues related to temporal fluctuations and instabilities of the single-layer (static) community detection process. We built on this existing work and leverage their approach to track popular temporal hashtag communities over time, which then form long-term topical trends.

## 4 Analysis and Evaluation

To illustrate the long-term trend detection method described in Section 3, it is applied to the political Twitter dataset as outlined in Section 3.1. Extracted hashtag co-occurrences are aggregated into monthly snapshots. For a global description of a trend, independent of time, the aggregated network as described in Section 3.2 is leveraged. As described in Section 3.4, the Leiden algorithm [TWV19] is used for the community detection step. We use modularity as the objective function along with a resolution parameter of 1,  $\beta = 0.01$  and 1000 iterations. Edge weights as outlined in Section 3.3 are taken into account. The algorithm is applied 10 times, and only the clustering that leads to the highest modularity score is taken to define the communities of hashtags, i.e., topics. Of course, due to the built-in randomness, repeated runs do not always lead to the exact same results but slight variations might occur. Per community, the induced subgraph of the 10 nodes, i.e., hashtags, with the highest PageRank scores [Pa99] is taken to represent a trend. Trend networks consist of those hashtags as nodes and their weighted interactions. As many communities contain hashtags that are either used for only a short time on social media or are very specific, we focus on the set of the 25 most central nodes, according to their PageRank, and link communities according to the similarity between those sets. For this, we leverage the method proposed by Lorenz et al. [Lo17] as described in Section 3.5. Four months are used as memory for the matching procedure to also link communities with temporal fluctuations and focus on the long-term prevalence of a trend.

In the following Section 4.1, we present analysis results covering the prevalence of trends over time, their evolution in Section 4.2, and temporal interactions in Section 4.3. Finally, results are evaluated in Section 4.4.

### 4.1 Prevalence of trends

As topics are tracked over time, their prevalence and popularity can be investigated with a focus on their temporal development. Not all topics might be equally prevalent at a given

point in time, nor might they be occurring across all time windows. Also, the popularity of an individual topic might change significantly over time. Figure 2 shows a temporal heatmap of the trend scores as outlined in Section 3.5. Trend scores are normalized on a trend basis, meaning that a value of 1 indicates the maximum of reached popularity for an individual trend. The heatmap shows the 10 trends with the overall highest trend scores and visualizes their development over the 18 months of the entire dataset.

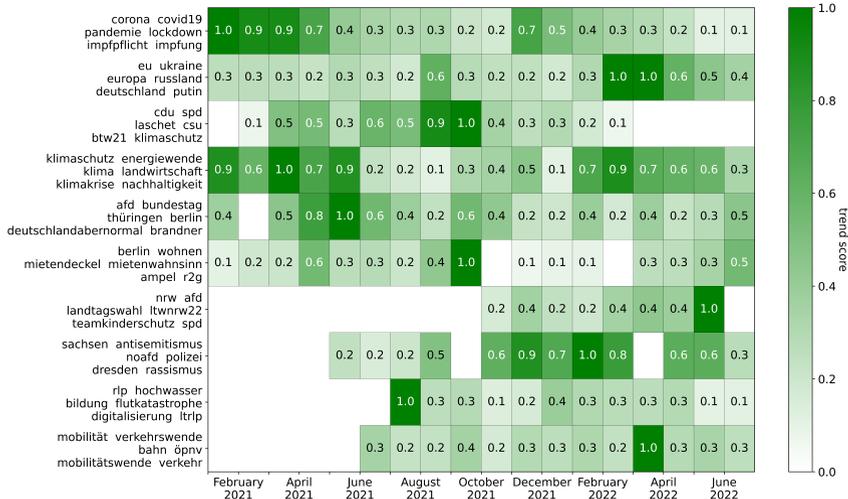


Fig. 2: Temporal heatmap of trend scores

First of all, it has to be noted that some trends, such as the one related to foreign policy and the European Union, are present across the entire time span whereas, for others, gaps in their prevalence over time become visible. That those gaps are occurring in the trend detection results confirms that the used method is indeed capable of handling temporal fluctuations. The topic is tracked over time even though it might not be detected in all intermediate snapshots. In contrast, some trends do not show gaps but are only present for a limited time span. As further described in Section 4.4, those trends are often related to some sort of event, e.g., the flood in the Ahr region. During the occurrence of that event, the trend’s popularity is often at its high. All trend developments show periods of higher and lower prevalence. As an example, the COVID-19-related long-term trend is most prevalent during the spring and winter of 2021, which might be due to a more tense pandemic situation during those periods. Further, some trends do peak at approximately the same time. Of course, one cannot conclude any causality or correlation from that but at least the heatmap makes such patterns visible. Exemplary of this are the peaks of the trends related to the Russian invasion of Ukraine, which also triggered an ongoing media discussion about public transportation (“mobilität”, “verkehrswende”) and renewable energy (“klimaschutz”, “energiewende”).

## 4.2 Temporal evolution

Topical trends do usually not consist of only a single keyword but are instead described by multiple aspects. With the proposed trend networks those aspects, represented by hashtags, and their interrelations are intuitively visualized. More interestingly, by tracking them over time the temporal changes in the topical trends can be analyzed. As an example, see Figure 3 that shows the trend networks related to the COVID-19 pandemic, as indicated by the respective hashtags, for the two time periods of January and November 2021. For the graph layout, the igraph [CN+06] implementation of the Fruchterman and Reingold [FR91] algorithm is used. Even though some hashtags can be found in both networks, e.g., “corona” and “pandemie”, other aspects and their importance change over time, e.g., “lockdown” and “impfstoff” vs. “impfpflicht” and “2g”. Also, it seems as for this trend, hashtags are a lot more interrelated during November 2021 as more edges in the network show. Represented by their weighting, those edges also indicate relationships of different strengths.

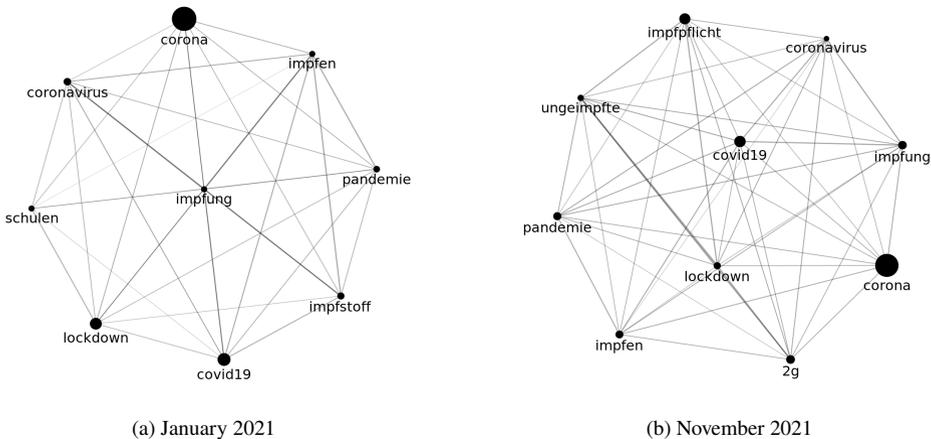


Fig. 3: Trend networks related to the COVID-19 pandemic covering different time periods

## 4.3 Trend interrelation

Chae and Park [CP18] already highlight the importance of topic interrelations. We go in the same direction and analyze *temporal* interrelations between topics. Topics do not co-exist independently of each other, but might instead be merged over time or at least become more or less interrelated.

Figure 4 visualizes the temporal interrelations between tracked trends for the time period of February until March 2022. The veins of the alluvial diagram [RB10] represent the flow of nodes between two communities and therefore, also the interrelation between topics across time. For the most part, topics seem to be quite stable as the majority of nodes stays

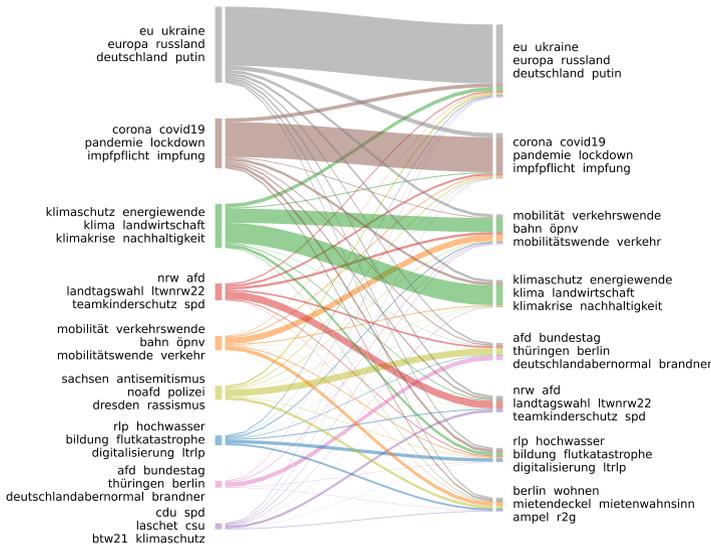


Fig. 4: Alluvial diagram visualizing the temporal interrelation of trends

within the same community. Nevertheless, some topics, e.g., the one related to climate protection, also influence multiple other ones, and nodes of these communities move to other topics. Most notably, a large portion of the climate protection topic shifts to the public transportation topic. To quantify these observations, 114 hashtags stay in the community, whereas 81 shift to the public transportation-related topic. Additionally, 21 shift to the foreign policy topic and 17 go to the one covering the Ahr flooding (see Section 4.4). Those results indicate a context switch of certain topical aspects as they become relevant for other trends as well.

#### 4.4 Evaluation

To confirm that computed trends are actually meaningful, we leverage an event-based evaluation and manually check if detected trends are related to real-world events. For the top 10 most prevalent trends (see Figure 2), the time frame of their highest popularity is taken as prediction and related events are checked for their temporal occurrence as kind of ground truth. In a subsequent step, the trend peak and the temporal occurrence of the related event are then compared and checked for accordance.

Table 1 shows that half of the top 10 long-term trends can be related to events, such as the COVID-19 pandemic or the Russian invasion of Ukraine. Popularity peaks of these trends are in close temporal proximity to the occurrence of the related events. We argue that for

Tab. 1: Long-term trends and related events

|    | Hashtags   | Peak                    | Event   | Reference (accessed 28-12-22)   |
|----|--|-------------------------|---|---|
| 1  | corona, covid19, pandemie, lockdown, impfpflicht, impfung                    | January 2021            | COVID-19 pandemic (17 November 2019 – present)                | <a href="https://en.wikipedia.org/wiki/COVID-19_pandemic">https://en.wikipedia.org/wiki/COVID-19_pandemic</a>   |
| 2  | eu, ukraine, europa, russland, deutschland, putin                            | February and March 2022 | 2022 Russian invasion of Ukraine (24 February 2022 – present) | <a href="https://en.wikipedia.org/wiki/2022_Russian_invasion_of_Ukraine">https://en.wikipedia.org/wiki/2022_Russian_invasion_of_Ukraine</a>   |
| 3  | cd, spd, laschet, csu, btw21, Klimaschutz                                    | September 2021          | 2021 German federal election (26 September 2021)              | <a href="https://en.wikipedia.org/wiki/2021_German_federal_election">https://en.wikipedia.org/wiki/2021_German_federal_election</a>   |
| 4  | Klimaschutz, energiewende, klima, landwirtschaft, klimakrise, nachhaltigkeit | March 2021              |   |   |
| 5  | afd, bundestag, thüringen, berlin, deutschlandabernormal, brandner           | May 2021                |   |   |
| 6  | berlin, wohnen, mietendeckel, mietenwahnsinn, ampel, r2g                     | September 2021          |   |   |
| 7  | nrw, afd, landtagwahl, ltw nrw22, teamkinderschutz, spd                      | May 2022                | 2022 North Rhine-Westphalia state election (15 May 2022)      | <a href="https://en.wikipedia.org/wiki/2022_North_Rhine-Westphalia_state_election">https://en.wikipedia.org/wiki/2022_North_Rhine-Westphalia_state_election</a>   |
| 8  | sachsen, antisemitismus, noafd, polizei, dresden, rassismus                  | January 2022            |   |   |
| 9  | rlp, hochwasser, bildung, flutkatastrophe, digitalisierung, lirlp            | July 2021               | Flooding of Ahr and Eifel region in Germany (15 July 2021)    | <a href="https://www.dw.com/en/flooding-in-germany-before-and-after-images-from-the-ahr-and-eifel-regions/a-58299008">https://www.dw.com/en/flooding-in-germany-before-and-after-images-from-the-ahr-and-eifel-regions/a-58299008</a> |
| 10 | mobilität, verkehrswende, bahn, öpnv, mobilitätswende, verkehr               | March 2022              |   |   |

the other trends as well meaningful descriptions can be found, like “climate protection” for trend 4, “AfD party” for trend 5, “housing market” for trend 6, “discrimination” for trend 8 and “public transportation” for trend 10. Nevertheless, those trends are not directly linked to real-world events. Together, the event-referenced and manually labelled trends prove good functionality of our long-term trend detection method.

## 5 Conclusion and Future Work

This work tackles the issue of detecting long-term prevalent topics, hidden in the large volume of short-lived news media. Based on methods known from the field of temporal network analysis and community evolution, an approach to detect such long-term trends is presented. A case study based on German political Twitter data proves that actually meaningful trends are detected. For a lot of the top trends, related real-world events can be identified, as shown in Section 4.4. Future work might target more extensive evaluation procedures and additional quantitative metrics to describe the long-term evolution of trends. Also, the current trend detection approach could be extended by a more sophisticated semantic topic model.

**Acknowledgements:** We thank the Klaus Tschira Foundation for funding this research in the framework of the EPINetz project: <https://epinetz.de>.

## References

- [ABP14] Alstott, J.; Bullmore, E.; Plenz, D.: powerlaw: a Python package for analysis of heavy-tailed distributions. *PLoS one* 9/1, e85777, 2014.
- [AH19] Annamradnejad, I.; Habibi, J.: A comprehensive analysis of twitter trending topics. In: 2019 5th International Conference on Web Research (ICWR). IEEE, pp. 22–27, 2019.
- [As11] Asur, S.; Huberman, B. A.; Szabo, G.; Wang, C.: Trends in social media: Persistence and decay. In: Proceedings of the International AAAI Conference on Web and Social Media. Vol. 5. 1, pp. 434–437, 2011.
- [BAE11] Budak, C.; Agrawal, D.; El Abbadi, A.: Structural trend analysis for online social networks. *Proceedings of the VLDB Endowment* 4/10, pp. 646–656, 2011.
- [Bh12] Bhulai, S.; Kampstra, P.; Kooiman, L.; Koole, G.; Deurloo, M.; Kok, B.: Trend visualization on Twitter: what’s hot and what’s not? In: 1st International Conference on Data Analytics. Pp. 43–48, 2012.
- [Bi18] Bianconi, G.: *Multilayer networks: structure and function*. Oxford university press, 2018.
- [CC08] Cryer, J. D.; Chan, K.-S.: *Time series analysis: with applications in R*. Springer, 2008.
- [Ch11] Chadwick, A.: The political information cycle in a hybrid news system: The British prime minister and the “Bullyinggate” affair. *The International Journal of Press/Politics* 16/1, pp. 3–29, 2011.
- [CN+06] Csardi, G.; Nepusz, T., et al.: The igraph software package for complex network research. *InterJournal, complex systems* 1695/5, pp. 1–9, 2006.
- [CP18] Chae, B.; Park, E.: Corporate social responsibility (CSR): A survey of topics and trends using Twitter data and topic modeling. *Sustainability* 10/7, p. 2231, 2018.
- [FR91] Fruchterman, T. M.; Reingold, E. M.: Graph drawing by force-directed placement. *Software: Practice and experience* 21/11, pp. 1129–1164, 1991.
- [Ha16] Hashavit, A.; Levin, R.; Guy, I.; Kutiel, G.: Effective trend detection within a dynamic search context. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. Pp. 817–820, 2016.
- [Kh21] Khan, H. U.; Nasir, S.; Nasim, K.; Shabbir, D.; Mahmood, A.: Twitter trends: A ranking algorithm analysis on real time data. *Expert Systems with Applications* 164/, p. 113990, 2021.

- [Kö22] König, T.; Schünemann, W. J.; Brand, A.; Freyberg, J.; Gertz, M.: The EPINetz Twitter Politicians Dataset 2021. A New Resource for the Study of the German Twittersphere and Its Application for the 2021 Federal Elections. *Politische Vierteljahresschrift*, pp. 1–19, 2022.
- [Le15] Le Masurier, M.: What is slow journalism? *Journalism practice* 9/2, pp. 138–152, 2015.
- [Lo17] Lorenz, P.; Wolf, F.; Braun, J.; Djurdjevic Conrad, N.; Hövel, P.: Capturing the dynamics of hashtag-communities. In: *International Conference on Complex Networks and their Applications*. Springer, pp. 401–413, 2017.
- [Ma20] Majdabadi, Z.; Sabeti, B.; Golazizian, P.; Asli, S. A. A.; Momenzadeh, O.; Fahmi, R.: Twitter Trend Extraction: A Graph-based Approach for Tweet and Hashtag Ranking, Utilizing No-Hashtag Tweets. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Pp. 6213–6219, 2020.
- [Ne05] Newman, M. E.: Power laws, Pareto distributions and Zipf’s law. *Contemporary physics* 46/5, pp. 323–351, 2005.
- [Pa99] Page, L.; Brin, S.; Motwani, R.; Winograd, T.: The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab, 1999.
- [RB10] Rosvall, M.; Bergstrom, C. T.: Mapping change in large networks. *PloS one* 5/1, e8694, 2010.
- [RN11] Role, F.; Nadif, M.: Handling the impact of low frequency events on co-occurrence based measures of word similarity. In: *Proceedings of the international conference on Knowledge Discovery and Information Retrieval (KDIR-2011)*. Scitepress. Pp. 218–223, 2011.
- [Tw] Twitter, Inc.: Twitter Trends FAQ – trending hashtags and topics, <https://help.twitter.com/en/using-twitter/twitter-trending-faqs>, Accessed: 28-12-22.
- [TWV19] Traag, V. A.; Waltman, L.; Van Eck, N. J.: From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9/1, pp. 1–12, 2019.
- [Zu15] Zubiaga, A.; Spina, D.; Martínez, R.; Fresno, V.: Real-time classification of twitter trends. *Journal of the Association for Information Science and Technology* 66/3, pp. 462–473, 2015.

## Session 4



# Duplicate Table Detection with Xash

Maximilian Koch,<sup>1</sup> Mahdi Esmailoghli,<sup>2</sup> Sören Auer,<sup>3</sup> Ziawasch Abedjan<sup>4</sup>

**Abstract:** Data lakes are typically lightly curated and as such prone to data quality problems and inconsistencies. In particular, duplicate tables are common in most repositories. The goal of duplicate table detection is to identify those tables that display the same data. Comparing tables is generally quite expensive as the order of rows and columns might differ for otherwise identical tables. In this paper, we explore the application of Xash, a hash function previously proposed for the discovery of multi-column join candidates, for the use case of duplicate table detection. With Xash, it is possible to generate a so-called super key, which serves like a bloom filter and instantly identifies the existence of particular cell values. We show that using Xash it is possible to speed up the duplicate table detection process significantly. In comparison to SimHash and other competing hash functions, Xash results in fewer false positive candidates.

**Keywords:** data discovery; data lakes; duplicate table detection

## 1 Introduction

The accelerating decentralized creation and publishing of data as well as the need for integration of such sources has led to a new wave of research on data market places [FSF20] and data lakes [Ar20]. Yet, these centralized data repositories have to deal with the distributed nature of data acquisition and the resulting data quality problems, one of which is duplicate data artifacts. An example of this is the Open Research Knowledge Graph (ORKG) project [Au20; Ja19]. On the ORKG platform, users can categorize and describe contributions from research papers and create additional properties to make them comparable and searchable in a structured form. Out of 524 tabular comparisons in the ORKG dataset, 48 are duplicates (9%). Figure 1 shows an example of two manually created comparison tables from the ORKG sharing duplicate rows [Te22]. As shown in Figure 1, two generated tables on a specific political science topic contain identical content derived from different literature. The discovery of such identical artifacts is useful. However, the attribute labels are rather misleading and the order of rows and columns is different. In a different real-world dataset, the DWTC corpus, containing 174M tables, there are 49M duplicates (28%) [Eb15a; Eb15b].

---

<sup>1</sup> Leibniz Universität Hannover, Germany koch@dbs.uni-hannover.de

<sup>2</sup> Leibniz Universität Hannover, L3S, Germany esmailoghli@dbs.uni-hannover.de

<sup>3</sup> TIB, Germany auer@tib.eu

<sup>4</sup> Leibniz Universität Hannover, L3S, Germany abedjan@dbs.uni-hannover.de

<sup>5</sup> Screenshots taken from <https://orkg.org/comparison/R110188/> and <https://orkg.org/comparison/R110245/>

|                      |  |   |   |
|----------------------|--|---|---|
| Properties           | FROM OTTOMAN TO REPUBLIC CENTER-PERIPHERY ANALYSIS IN TURKISH POLITICAL CULTURE AND BUREAUCRACY<br>2013 - Contribution 1 | Participating in the design: constitution-making in South Africa<br>1996 - Contribution 1 | Constitution Making and Democratization in Kenya (2000–2005)<br>2007 - Contribution 1 |
| has_research_problem | Constitution-Making Process  | Constitution-Making Process   | Constitution-Making Process   |
| has_method           | Unanimity Principle  | Qualified Majority  | Qualified Majority  |
| institution          | Constitutional Reconciliation Commission (In National Assembly)  | Constitutional Assembly   | National Assembly   |
| results              | Failure  | Successful  | Successful  |
| country              | Turkey   | South Africa  | Kenya   |

|                      |   |   |  |
|----------------------|---|---|--|
| Properties           | Participating in the design: constitution-making in South Africa<br>1996 - Contribution 1 | Constitution Making and Democratization in Kenya (2000–2005)<br>2007 - Contribution 1 | A FAILED PROCESS OF MAKING A NEW CONSTITUTION-AN EVALUATION ON THE CONSTITUTIONAL RECONCILIATION COMMISSION<br>2016 - Contribution 1 |
| has_research_problem | Constitution-Making Process   | Constitution-Making Process   | Constitution-Making Process  |
| has_method           | Qualified Majority  | Qualified Majority  | Unanimity Principle  |
| institution          | Constitutional Assembly   | National Assembly   | Constitutional Reconciliation Commission (In National Assembly)  |
| results              | Successful  | Successful  | Failure  |
| country              | South Africa  | Kenya   | Turkey   |

Fig. 1: Duplicate table example in the Open Research Knowledge Graph.<sup>5</sup>

Duplicate detection has been the focus of research for several decades. Most existing work focuses on record linkage, i.e., finding records that represent the same real-world entity [Ch12a; Ch12b; Li20; LSR21; Th20]. Another line of research has dealt with the identification of (near)-duplicate documents [CGS03; Jo72]. In general, the fundamental challenges in duplicate detection are that pairwise comparisons of all considered entities are computationally expensive, i.e.,  $O(n^2)$  for  $n$  entities, and that effective similarity metrics are necessary to capture non-exact matches. To reduce the number of table-to-table comparisons one has to resort to pre-filtering techniques that apriori discard non-matching pairs.

In this paper, we focus on the discovery of *duplicate tables* within a data lake. We consider two modes of duplicate table detection:

1. *Duplicate table retrieval*: Given a user table, the goal is to identify the existence of all tables that match or contain the given table.
2. *Lake de-duplication*: Given a repository of relational tables, the goal is to identify all duplicate tables within a lake.

We consider two tables  $T_1$  and  $T_2$  to be duplicates if a permutation of columns in  $T_1$  exists  $T_1^P$  so that  $T_1^P$  and  $T_2$  contain the same set of tuples. While this definition simplifies the problem of duplicate table detection by excluding fuzzy matches, there are still runtime bottlenecks. In table retrieval, the initial identification of candidate tables requires the retrieval of tables with matching rows. In lake de-duplication, we require a blocking technique similar to what has been proposed in the duplicate detection literature [Ch12a; Fi15; Ga22]. After which again table-to-table matches have to be considered. As we exclude fuzzy matches, blocking can be as simple as grouping tables based on the number of rows and columns and then hashing them via Simhash into smaller buckets.

The overarching challenge in both detection modes is that in order to verify the match of two tables one has to compare the entire content of both tables after aligning the columns, which is a computationally expensive process when carried out in a naïve manner. A naïve approach to compare two such tables is to first sort the values inside each row alphabetically (horizontally) and then hash the rows into matching buckets. Note that sorting based on column labels might be misleading as duplicate tables might differ in the actual column labels. To circumvent this, the column position is stored for each unsorted row. For  $m$  columns and  $k$  rows, this results in  $O(2 \cdot m \cdot \log(m) \cdot k + 3 \cdot k)$ , i.e., sortation of  $m$  values of all  $k$  rows and the application of a collision-free hash function to match the sorted rows into  $k$  different buckets. A final pass is necessary to verify that the original column order is consistent in all matched rows. For large number of tables, this approach will be prohibitively expensive. Furthermore, in a retrieval scenario, where tables are retrieved via an inverted index, one has to first retrieve a set of candidate tables that partially match on a chosen query column.

In this work, we propose hash-based solutions for fast comparison of duplicate candidates as well as fast discovery of candidates from a large data lake. Our proposed solution is inspired by a previously introduced hash function framework MATE [EQA22], which was designed for efficient discovery of multi-column join candidates. MATE leverages a hash function XASH to mask the existence of each row value of a row within a unified bit string and applies a bloom-filter-inspired approach with a so-called *super key* to discard non-joinable candidates. Doing so speeds up the identification of joinable tables by orders of magnitude.

Inspired by the capabilities of XASH, we explored its application for the table de-duplication case, which in a sense translates to joining two tables on all attributes of both tables. When searching for duplicates, the bloom-filter-like structure can be used to rule out non-duplicate rows without the need of reordering the columns and knowing the schema of the tables.

If the hash values of two rows are not equal, the rows are not equal and do not need to be checked in more detail. Otherwise, it could be evidence of a duplicate table relationship.

In this paper, we explore the application of XASH and other similar hash functions for the two detection modes described above: lake de-duplication and duplicate table retrieval. We describe the workflow of each detection process and possible optimizations when dealing with tables. We compare the pruning power to other hash functions and discuss situations where the application of the filtering with XASH is beneficial over direct comparisons.

## 2 Related Work

Duplicate table detection is related to several lines of research, such as duplicate web page detection, entity resolution, fuzzy joins, and data discovery.

### 2.1 Duplicate document detection

Duplicate document detection has been vastly studied to enhance the effectiveness of search engines and diversify search results by finding duplicate web pages and dropping these results from the search output [CGS03; He06]. Web pages are mainly comprised of HTML content and are treated as text documents instead of structured data such as tables. Because of this, the tables in web pages are also treated as pure text [CCB02]. These duplicate detection approaches apply feature generation techniques, such as shingling [Br97], sentence extraction [Ku17], and stop word removal, and tokenization [TSP08].

These techniques do not consider the highly structured nature of relational tables with numerical and distinct columns [Br97; TSP08]. Generally, any approach based on the approximation of table content through stop word removal and tokenization can serve the table grouping step and is orthogonal to our proposed techniques. Once a group of candidate tables is retrieved, the filtering with the super key can take place.

### 2.2 Entity resolution

In entity resolution, the goal is to discover data records that represent the same entity in the real world [Ch12b; Ch21; KTR10; Si22]. Entity resolution methods leverage matching functions that employ similarity metrics to create clusters of table rows that are candidates to be duplicates [Si22]. To reduce the number of pairwise comparisons, entity resolution typically applies blocking methods for fast discovery and exclusion of non-matching pairs [Ch12a; Fi15].

The main difference between entity resolution and duplicate table detection is that the former only focuses on the similarity of the rows [KPN20]. For two tables to be duplicates, all rows

of the two tables should find a pendant duplicate. Furthermore, the schema should be very similar. Entity resolution, cannot directly apply to table-level discovery. It is challenging to systematically decide on two tables being duplicates based on the similarity score of their rows. Therefore, we focus on the case of exact table duplicates where only reordering of rows and columns is allowed, which is already hard enough for a large set of tables.

### 2.3 Fuzzy Joins

Fuzzy joins, i.e., non-equi or similarity joins, aim to discover the joinable rows from different tables, where the rows have similar keys [WLF11; Yu16]. Xiao et al. aim to discover near-duplicate tables using join discovery, i.e., set similarity search [Xi11]. They leverage the positioning filter to efficiently prune the search space and discover joinable tables for a given table and a join column. However, they only consider single-attribute joins. Thus, joinable tables are not necessarily duplicates. Because the similarity join discovery algorithms only discover the similarity based on one key column per table, these approaches can only serve at the initial candidate retrieval stage.

### 2.4 Data Discovery

Other data discovery approaches, such as union discovery [Na18] algorithms, also focus on partial similarities. For instance, two tables with a very high unionability score might not even have a single overlapping value, which means that they are not duplicates while they might be unionable.

## 3 Fundamentals

The core component of our table duplication approach is the so-called *super key*, which is a bit string aggregated from multiple applications of the hash function `XASH` developed for multi-attribute join table discovery [EQA22]. In this section, we briefly review important characteristics of `XASH` to motivate its suitability for duplicate table detection and describe the inverted index structure that maintains the generated super keys.

### 3.1 XASH Design Goals

The design goal for `XASH` was to hash each row of a table in a way that within a constant operation it is possible to identify whether the row contains a specific value combination or not. Thus, it has the core properties of a bloom filter as it does not lead to any false negatives and is highly effective in differentiating non-equal but similar rows, regardless of the column order.

Furthermore,  $X_{ASH}$  is designed in a way that the hash of any set of values is masked by the hash of any of its potential supersets. That is why it can serve for arbitrary multi-attribute join keys.

To differentiate row values that are non-identical,  $X_{ASH}$  captures differentiating features of each value. For each row value,  $X_{ASH}$  encodes the least frequently occurring characters, the location of these characters, and the length of the value. To further differentiate non-identical rows that by accident end up with the same set of rare characters being encoded. Given a value of length  $l$ ,  $X_{ASH}$  shifts the resulting code  $l$  bits to the left. Thus for two values to have the same hash, length, rare character distribution and positions must match.

Given a hash string of  $n$  bits,  $X_{ASH}$  divides the bit string into several segments, one segment per eligible character and a remaining segment to encode the length. Typically the hash size must be chosen in a way that each eligible character (space, 0-9, lower-case a-z) can be covered by at least a 1-bit segment. In practice, with a hash size of 128 and larger, the segments can be larger, which makes it possible to encode the relative location of the occurrence of the encoded characters within the corresponding segment.

The remaining bits that equal to the remainder of the division of hash size and a number of eligible characters are used to encode the length of each encoded row value. The encoding of the length calculates the string length modulo the number of available bits for the length segment. Thus only one bit needs to be set to encode the length.

### 3.2 $X_{ASH}$ -Aggregation per row

After generating the  $X_{ASH}$  for each value of a row, all hash results are aggregated via a bitwise OR operation. The result of this aggregation is a so-called *super key*.

The super key is generated while indexing the corpus and can now be probed like a bloom-filter to check whether a value combination is included or not. Given the aggregated hash value  $h_c$  of a candidate value combination  $C$  and the super key  $h_r$  of a row  $r$ , the operation  $h_c \vee h_r$  should result in  $h_r$  if there is a chance that the candidate value combination is contained. Similar to a bloom-filter,  $h_c \vee h_r \neq h_r$  will always correctly identify that the  $C$  is not contained in  $r$ , however, might be inaccurate if  $h_c \vee h_r = h_r$ . In the latter case, additional verification is necessary. Experiments and proofs in prior work show that  $X_{ASH}$  leads to significantly fewer false positives than the state-of-the-art [EQA22].

**Example.** To illustrate the hash generation using  $X_{ASH}$ , we use the following example from the World Bank’s current GDP dataset [Wo22] as shown in Table 1.

Figure 2 visualizes the process of  $X_{ASH}$  generation and the merge into the super key. The hash size is 128 bits. To select the least frequent characters, each row value is converted to lower case and only the characters a-z, 0-9, and space are kept. For each row value, the

Tab. 1: Example data: entry from the World Bank’s current GDP dataset.

| Country Name                                  | Country Code | 2020             |
|---|--------------|------------------|
| Europe & Central Asia (excluding high income) | ECA          | 3222403620453.33 |

three least frequent characters are selected (marked bold): europe central asia excluding high income); eca; 32224**0362045**333.

In the hash value, there are three bits available per character. If the average location of one of the least frequent characters is in the first third of the row value, the first bit will be set to 1, the second bit if it is in the second third, and the third bit if it is in the third third. For example, **p** in “europe central asia excluding high income” appears in the first third of the entire value. Thus, the first bit of the **p**-segment is set to 1.

With 128 bits and 37 eligible characters,  $128 - 37 * 3 = 17$  bits remain for the length segment. To encode the length of the first row value,  $41 \bmod 17 = 7$  (41 is the length of the row value, 17 the segment length) is calculated, which means that the seventh bit in the length segment will be set to 1.

Lastly, all set bits are shifted left by 41 while the overflow is added from the right, ignoring the length segment.

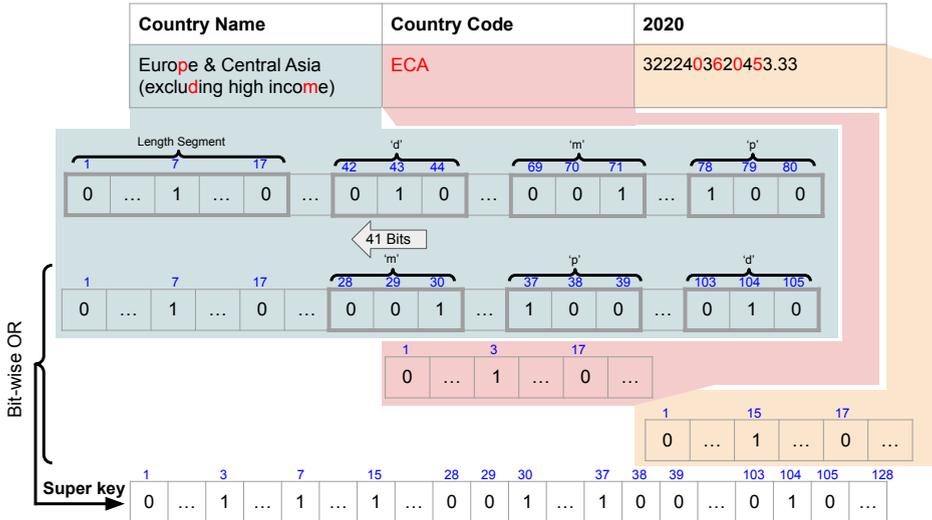


Fig. 2: Example of XASH super key generation

### 3.3 XASH for duplicate detection

While the probing operation of XASH for join discovery probes for containment, its application for duplicate row detection can be more strict as the containment has to go both ways. Thus, for two rows to be duplicates both hash values have to be exactly the same.

For two tables to be duplicates of each other there has to be a permutation of columns, so that all tuples of one table are contained in the other and vice versa. For practical reasons we relax this duplicate definition as follows:

- We ignore the duplication of rows within a single table and consider two tables to be duplicates as long as each unique row exists in both tables.
- In our implementation, we also enable the discovery of tables that fully contain the rows of another table, meaning that one table has more rows than the other. Note, that the number of columns has to be equal. As we will see in Section 4, this decision does not affect the way XASH is applied for filtering.

### 3.4 Inverted Index

To achieve fast query results when interacting with the lake, data discovery systems usually leverage one form of an inverted index [Ab16; EQA22; Fe18; Zh19]. An inverted index is well-known in the context of information retrieval and maps tokens to containers, such as documents or tables [GF98]. In the context of data lakes, the containers are the corresponding tables, rows, and columns. As we rely on the MATE framework, we use their inverted index, where an entry inside the index consists of the mapping of the tokenized value to the corresponding table, row, and column IDs [Ab16; EQA22].

In the example shown in Figure 1, the row values 'Turkey', 'South Africa', and 'Kenya' appear in both tables whom for simplicity we assign the ids 1 and 2. With that the content of the inverted index would be as follows:

*Turkey*  $\rightarrow \{1, 2\}$

*South Africa*  $\rightarrow \{1, 2\}$

*Kenya*  $\rightarrow \{1, 2\}$

The schema of the index includes one row of one table from Figure 1 and the super key column is shown in Table 2.

In addition to the aforementioned mapping, the MATE framework also maps the super key to each row value so that information about each row is readily available when probing for one of the row values [EQA22]. Additionally, an index is created on the super key column, to be able to retrieve results from the table when using the super key as a filtering criteria.

Tab. 2: Schema of the inverted index in the database

| tokenized    | tableid | colid | rowid | super_key |
|--------------|---------|-------|-------|-----------|
| Turkey       | 1       | 0     | 0     | ...       |
| South Africa | 1       | 1     | 0     | ...       |
| Kenya        | 1       | 2     | 0     | ...       |

**Algorithm 1:** Duplicate table retrieval

---

```

1 Inputs: user_table
2 duplicate_tables = []
3 super_key_mapping = csvToSuperKeyRowIdMap(user_table) /* Map super key to row ids */
4 input_rows = super_key_mapping.values()
5 input_superkeys = super_key_mapping.keys()
6 rows = getDbRowsWithSameSuperKey(input_superkeys) /* Get all rows from the database, that have one of
   the super keys from the input rows */
7 foreach row in rows do
8   input_rows_candidates = super_key_mapping.get(row.superkey) /* Get all rows from the input table, that
   have the same super key as the current db row using hash join */
9   foreach input_candidate in input_rows_candidates do
10    /* check the correspondence of rows, yet make sure that column positions remain consistent across
   matched rows */
11    if verifyRows(input_candidate, row) then
12      table_id_to_rows.add(retrieveTableId(row), retrieveRowId(row));
13      table_id_to_rows_input.add(retrieveTableId(row), retrieveInputId(row));
14 foreach (tableid, rowids) in table_id_to_rows do
15   /* It is checked if there are duplicate tables, based on the detected duplicate rows for each db table */
   duplicate_tables.add(getDuplicateTables(tableid, rowids, table_id_to_rows_input[tableid],
   table_id_to_rows[tableid]));
16 return duplicate_tables;

```

---

## 4 Duplicate Table Detection

In this section, we describe how a hash-based index can be used for duplicate table retrieval as an online process for a given user table and as an offline process to de-duplicate a group (a block) of tables. For both applications, we present algorithms that leverage the super keys based on XASH.

### 4.1 Duplicate Table Retrieval

In the first scenario, the user provides a table and is searching for all possible duplicates or subsuming tables inside the data lake at hand.

Given an inverted index as suggested in Section 3, the naïve approach would use the content of one column to fetch all tables that contain all values of that particular column. Then all remaining columns of the fetched tables would be loaded so that the remaining columns of the input table can be verified against each candidate table.

To reduce the runtime of this step and the number of string comparisons, one can leverage `XASH` as a filter. This requires us to hash all rows of the input table using `XASH` and to retrieve the super keys of lake tables that were generated during the indexing phase.

Algorithm 1 depicts the process in more detail. Given the user table, the algorithm first iterates through its rows to calculate the super key for each row and keeps them accessible for later probing.

After processing the input table, the approach retrieves candidate rows from the lake. Using the inverted index, a query is submitted to the data lake where to retrieve any row with a super key that appears in the input table (line 6). Each retrieved row from the data lake is compared with all rows from the input table having the same super key (line 11). This step verifies for two rows with the same super key whether they indeed contain the same row values, regardless of the order.

If an input row and a row from the lake match, the table ID and the row ID of the data lake table as well as the row ID of the input table are temporarily stored in `table_id_to_rows`(line 12) and `table_id_to_rows_input` (line 13).

After all of the rows passed the aforementioned checks, all tables that share any row with the input table are verified (line 15). Using the aforementioned table-to-row maps, tables that either contain all input table rows or are subsets of the input table are identified ( line 15). The `getDuplicateTables` function uses the `table_id_to_rows_input` map to check which of the rows of the input table occur in the database table and the `table_id_to_rows` map, to check which rows of the database table occur in the input table.

## 4.2 Lake De-Duplication

Finding duplicate tables in a data lake requires comparing all pairs of tables.

Typically, some sort of blocking has to be applied to reduce the number of pairwise table comparisons. For the sake of context, Figure 3 shows a conceptual pipeline for Lake De-Duplication including the blocking step. Generally, the blocking strategies have to be tailored to the type of duplicates we are after. If only exact duplicates with arbitrary row and column orders are considered, the blocking can already take the table dimensions into consideration. If fuzzy duplicates are of interest hashing approaches based on Simhash for near-duplicate document detection can be considered. In this paper, we consider the former situation and apply a very simple blocking technique that sorts out tables with equal dimensions. Our approach is orthogonal to blocking. Rather we show, that given a coherent group of tables, i.e., all tables that are in the same block or have the same row and column dimensions and share some general similarity, the existence of the super key significantly improves the overall pairwise comparisons.

All pairs of tables contained in a block must undergo pairwise comparisons. To this end, we join each pair of tables using hash join with `XASH` super keys and pass the joint table to the validation step. In this step, the joinable rows are validated to discover and drop the false positive rows, i.e., rows that have the same super keys but are not joinable. Ultimately, the tables are duplicates if the number of duplicate rows equals to the number of rows in the smallest table.

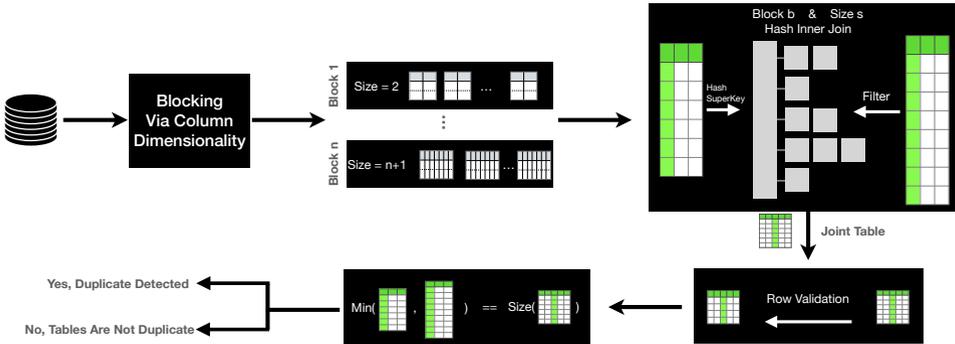


Fig. 3: Lake de-duplication pipeline.

Algorithm 2 shows the process of comparing two tables in detail. *CompareTables* receives two tables as input and returns as the result whether the tables are duplicates, or strictly contained in one direction. First, the smaller and the bigger tables are identified (Line 3). This helps us to create the hash table effectively and to find the candidate subset table. In the first loop, the algorithm (Lines 4-6) creates a hash table based on the super keys in the bigger table. The hash table is created based on the bigger table because in the filtering phase using the smaller table, we can also identify a one-directional subset relationship. Creating the dictionary based on the bigger table allows us to make sure that if even one super key in the smaller table does not exist in the dictionary, we can make sure that the two tables at hand are neither duplicates nor subsets of each other (Lines 9, 10). On the contrary, if a super key from the smaller table exists in the hash table (Line 13), we get the corresponding candidate rows from the larger table (Line 14) and verify them (Line 15). The *verifyRows* compares the actual cell values of rows with the same super key. For this purpose, it sorts the cell values of each row and checks whether the two rows are equal. To save runtime, each row is only sorted at most once and the sorted row is cached in case it is needed for later comparisons. For each matched row the function records the resulting column alignment. If two consecutive row matches result in different column alignments, the tables are considered as non-duplicates. If the rows are duplicates, they will be stored in the *matched\_rows* list (Line 16). If the size of *matched\_rows* equals the size of the smaller table, it means that the bigger table contains all the rows in the smaller table and there is a subset relationship (Line 17).

The proposed algorithm and pipeline lead to zero false negatives. Based on the definition of duplicates in this paper, two duplicate tables must have the same number of columns.

---

**Algorithm 2:** CompareTables

---

```
1 Inputs: =t1: table 1, t2: table 2,
2 matched_rows = []
3 smaller_table, bigger_table = getSmallerBiggerTable(t1,t2)
4 foreach row_t1 in bigger_table do
5   | super_key_t1 = bigger_table[row_t1].get_super_key()
6   | hashjoin_map[super_key_t1].append(row_t1)
7 foreach row_t2 in smaller_table do
8   | super_key_t2 = smaller_table[row_t2].get_super_key()
9   | if super_key_t2 not in hashjoin_map then
10  |   | break
11  | else
12  |   | row_t2_values = smaller_table[row_t2]
13  |   | foreach hit_row_t1 in hashjoin_map[super_key_t2] do
14  |   |   | row_t1_values = bigger_table[hit_row_t1]
15  |   |   | if verifyRows(row_t1_values, row_t2_values) then
16  |   |   |   | matched_rows.append(hit_row_t1, row_t2)
17 return ||matched_rows|| == ||smaller_table||
```

---

As our blocking method groups the tables based on their column dimensionality, i.e., the number of columns, there will be no misses in during blocking.

The bloom-filter-like property of XASH assures that any two rows that are duplicates will have the same super key. Therefore, no duplicate rows will be missed if their super keys do not match.

## 5 Experiments

We carried out a series of experiments to answer the following questions: (1) Can we significantly improve the runtime of duplicate table detection using the XASH filter? (2) How does modifying the XASH generation affect the number of false positives? (3) How do other hash functions perform for the same purpose?

### 5.1 Experimental Setup

We conduct experiments for both setups: duplicate table discovery and table group de-duplication.

### 5.1.1 Dataset for duplicate table discovery

To test the efficiency of our approach, we executed duplicate detection tasks on top of the DWTC dataset [Eb15a]. The corpus consists of 145,533,822 tables and is indexed as described in Section 3.

As the algorithm expects a table for input, we randomly selected 5 tables for each row and column number dimensions of 1, 10, 100 and 5, 10, 50, respectively. In all our experiments, we also retrieve tables with subset relationships (see Section 4). Turning this check to exact duplicates does not change the performance.

### 5.1.2 Dataset for lake de-duplication

To evaluate the effectiveness of the super key filter in a lake de-duplication scenario, we simulate the generation of groups where table-to-table comparisons have to take place at the row level. We sampled coherent groups of tables, i.e., with equal table dimensions, from the Wikipedia dataset<sup>6</sup> [Au07].

The Wikipedia dataset consists of 7,684,431 different tables, with 380,475,701 total entries in the inverted index described in Section 3. For the different test runs on this dataset, we sampled real groups of 1,000-50,000 tables to represent duplicate blocks.

### 5.1.3 Competitors

For both scenarios, we are interested in the number of false positives and the runtime in comparison to the naïve approach and other hash functions.

- **Naïve approach:** In this approach, duplicate rows are detected by checking if two rows are equal through sortation and step-wise comparison of the aligned row values.
- **XASH:** Two rows are compared by checking if the super keys generated using XASH are equal. The super keys consist of 128 bits, as proposed in the original paper [EQA22]. If the super keys of two rows are equal, the row values need to be checked the same way as described for the naïve approach, to rule out a false positive.
- **SimHash:** SimHash was developed for the purpose of finding similar content using hash values [MJS07]. Its application is similar to using XASH, with the difference that SimHash was used to generate the super keys.
- **CityHash:** CityHash was developed to generate hashes quickly, while still resulting in mostly unique values [PA22]. The application is similar to using XASH or SimHash.

<sup>6</sup> <https://databus.dbpedia.org/dbpedia/text/raw-tables>

- **MD5:** MD5 was chosen, as it is a broadly used hash function. Its application is similar to using XASH, SimHash, or CityHash.

All experiments were executed on a server with an AMD EPYC 7702P CPU with 64 cores/128 threads, 528 GB RAM, and 10 TB storage space. All code is implemented in Python 3.9.2.

We used PostgreSQL 13.7 to store the inverted index. The indexes map tokenized row values of the dataset tables to the corresponding table, row, and column ids as well as the row super keys. Further, there is an index on the tableids and a third one on the super keys. All code is available on GitHub: <https://github.com/LUH-DBS/XashDedup>.

## 5.2 Response time in duplicate table discovery

Table 3 displays the results of our main experiments to assess the efficiency of duplicate table retrieval under the presence of different hash functions for super keys. We report the runtime average for five tests per input dimension, i.e., ( $X$  rows and  $Y$  columns). Furthermore, we report the average percentage of false positives (FP %), which is calculated as  $\frac{FPs}{FPs+TPs}$  and measures the ratio of non-duplicate rows that wrongly passed a hash-based filter. In essence, a good filter has a low FP%.

The approach based on XASH has the lowest percentage of false positives across all input table dimensions, except for 10 rows / 50 columns. Note that although for 50 columns the FP rate is about the same rounded number of 100% for CityHash, SimHash and MD5 there are significant differences in the absolute numbers. Using XASH, on average 148k false positives passed the filter, while for other hash functions this number was 750k, 580k, and 632k, for CityHash, SimHash, and MD5 respectively. This difference is clearly reflected in the runtime. XASH yields the lowest average runtime by at least one order of magnitude, compared to the filters with the other hash functions. The experiment is repeated 5 times and the runtime superiority of XASH compared to the other hash functions is statistically significant within 99% confident interval. For 100 rows and 5 columns, the approach based on XASH clearly shows an advantage in terms of false positives and runtime: with only 3.4% of the passed rows being false, the runtime of 688ms is at least 57x faster than the competitors. For the (1,5) dimension SimHash has a slightly better average runtime despite the higher FP rate because in some experiments high similarity of rows inside the input table result in identical hash functions, which in turn reduces the retrieval effort for such input tables.

We make several further observations with regard to the influence of the input dimensions on the overall runtime and FP rate. First, the more columns the input table contains the higher is the overall false positive rate. This is because the hash functions are aggregates of row value hashes. Thus, the more values are hashed and ORed the higher will be the number of 1-bits, i.e., bits that turned to 1, and the more likely it will be that by chance the same

Tab. 3: Runtime, result size, and false positives (FP) **averaged** over 5 experiments per input dimension

|                  | 1 Row / 5 Cols    |             | 1 Row / 10 Cols    |             | 1 Row / 50 Cols   |             |
|------------------|-------------------|-------------|--------------------|-------------|-------------------|-------------|
| Average...       | Runtime (ms)      | FP %        | Runtime (ms)       | FP %        | Runtime (ms)      | FP %        |
| X <sub>ASH</sub> | 1,023             | <b>1.1</b>  | <b>320</b>         | <b>2.3</b>  | <b>713</b>        | <b>74.6</b> |
| CityHash         | 34,470            | 74.6        | 5,002              | 100.0       | 21,611            | 100.0       |
| SimHash          | <b>963</b>        | <b>1.1</b>  | 19,977             | 99.9        | 9,754             | 100.0       |
| MD5              | 16,038            | 50.0        | 58,353             | 100.0       | 21,856            | 100.0       |
| No Hash          | 2,639,586         | -           | 1,787,112          | -           | 1,823,862         | -           |
|                  | 10 Rows / 5 Cols  |             | 10 Rows / 10 Cols  |             | 10 Rows / 50 Cols |             |
| Average...       | Runtime (ms)      | FP %        | Runtime (ms)       | FP %        | Runtime (ms)      | FP %        |
| X <sub>ASH</sub> | <b>135</b>        | <b>23.4</b> | <b>150</b>         | <b>0.0</b>  | <b>10,440</b>     | 100         |
| CityHash         | 41,428            | 99.8        | 23,420             | 100.0       | 36,927            | 100         |
| SimHash          | 24,406            | 100         | 25,692             | 100.0       | 37,610            | 100         |
| MD5              | 8,464             | 98.6        | 23,238             | 100.0       | 370,843           | <b>78.4</b> |
| No Hash          | 403,565           | -           | 2,767,554          | -           | 1,495,390         | -           |
|                  | 100 Rows / 5 Cols |             | 100 Rows / 10 Cols |             |                   |             |
| Average...       | Runtime (ms)      | FP %        | Runtime (ms)       | FP %        |                   |             |
| X <sub>ASH</sub> | <b>688</b>        | <b>3.4</b>  | <b>397</b>         | <b>10.5</b> |                   |             |
| CityHash         | 39,707            | 99.9        | 79,712             | 100.0       |                   |             |
| SimHash          | 445,152           | 99.5        | 71,710             | 100.0       |                   |             |
| MD5              | 280,832           | 100.0       | 80,524             | 100.0       |                   |             |
| No Hash          | 2,156,850         | -           | 1,454,387          | -           |                   |             |

bits are turned to 1. There is an anomaly in the runtime with the (1,5) input datasets for X<sub>ASH</sub> as the runtime is higher than all other chosen dimensions except (10, 50). The reason is that there are too many actual duplicates that need to be processed for (1,5). For larger dimensions, the probability for duplicates and so true positives naturally decreases so that the number of FPs and the rate becomes more relevant. As expected, the approach with no hash function filter displays a significantly higher runtime than the hash-based approaches. We enforced a limit of 1M cells for the number of retrieved rows so that the naïve approach would be able to finish. Small deviations in the number of true positives are a result of a hard limit of the enforced limit. The runtime might be higher in cases where the limit is exceeded, i.e., the other hash functions.

### 5.3 Runtime Lake De-Duplication

As we make no claims on how to obtain the duplicate groups in lake de-duplication, we only present experiments on the pairwise comparison approach in randomly selected duplicate groups of 1,000 tables (7,284 rows; 54,434 row values, 96 duplicate tables tuples), 5,000 tables (33,937 rows; 233,807 row values, 2,528 duplicate tables tuples), and 10,000 tables (66,091 rows; 404,999 row values, 10,845 duplicate tables tuples). We repeat each experiment five times and report the average.

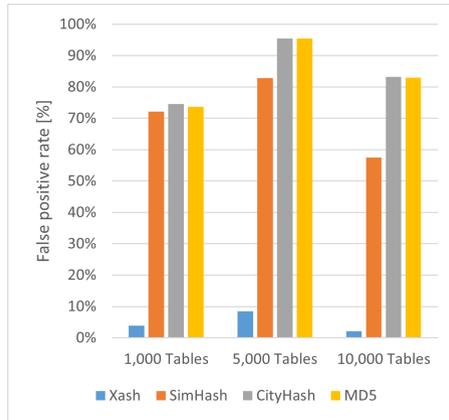


Fig. 4: Avg. false positive rates of table comparisons in duplicate groups

|          | 1,000 Tables       | 5,000 Tables       | 10,000 Tables      |
|----------|--------------------|--------------------|--------------------|
|          | Avg. runtime in ms | Avg. runtime in ms | Avg. runtime in ms |
| Xash     | <b>401</b>         | <b>9,229</b>       | <b>48,601</b>      |
| SimHash  | 424                | 9,935              | 50,920             |
| CityHash | 457                | 10,417             | 52,610             |
| MD5      | 458                | 10,509             | 52,738             |
| No Hash  | 7,669              | 203,769            | 716,738            |

Tab. 4: Runtime of table comparisons in duplicate groups

Figure 4 shows the false positive rate and Table 4 contains the runtime for the different hash functions and table groups.

Across all tested groups, XASH has the lowest percentage of false positives with a false positive rate below 2.5%, followed by SimHash with a minimum of 58% false positives, CityHash and MD5 with both around 74% false positives at minimum. Accordingly, XASH application results in the lowest runtime.

The number of false positives increases with the size of the groups for all approaches. Similarly, the runtime of each approach increases each time by an order of magnitude when increasing the group size from 1,000 to 5,000 and then to 10,000.

For 1,000 tables, 2,509,878 row comparisons were performed on average for 5 runs with the naïve comparison algorithm, which leads to a runtime of 7,669ms. Using the MD5 hash function reduces the number of comparisons to 15,797 similar to CityHash with 15,681. Despite the drastic reduction of comparisons by a factor of 160, the runtime is only reduced by a factor of 16, to 458ms for MD5 and 457ms for CityHash. This is due to the fact, that while using a super key eliminates some row comparisons, the super keys still need to be compared with each other using a hash join approach. Furthermore, the overhead of

retrieving tables becomes a more noticeable process. Using X<sub>ASH</sub>, there are on average only 67 false positives out of 1,734 rows, making its approach the fastest. X<sub>ASH</sub> outperforms SimHash significantly within the 99% confidence interval.

For groups of 5,000 and 10,000 tables, we see a similar relative performance relationship as observed for the group of 1,000 tables. Using X<sub>ASH</sub> results in the fastest execution time of 48,601ms for 265,067 rows in 10,000 tables, while SimHash, CityHash, and MD5 are slightly slower. Using SimHash leads to the second-best runtime with 50,920ms, but around 62x more false positives.

Using no hash functions and therefore comparing all rows with each other results in a runtime of 716,738ms and 270,718,502 row comparisons in the largest group. Using any hash function the runtime can be reduced by more than 93%, 92% with X<sub>ASH</sub>.

### 5.3.1 Varying the Hash Size

The hash functions usually require size parameters that specify the number of bits the returned hash value has. To examine the effect of the hash size, the *compareTables* algorithm is executed with different hash sizes of 64, 128, and 256 for X<sub>ASH</sub> and Cityhash. For SimHash, 64 and 128 bits were used, as there was no implementation for 256 bits available. The experiment was performed on the 10,000 tables group of the Wikipedia dataset.

Tab. 5: Runtime comparison: different hash sizes

|                      | Runtime in ms | FP        | SUM (FP+TP) |
|----------------------|---------------|-----------|-------------|
| X <sub>ASH</sub> 64  | 32,051        | 5,725     | 70,562      |
| X <sub>ASH</sub> 128 | 31,537        | 5,535     | 70,354      |
| X <sub>ASH</sub> 256 | 31,809        | 4,455     | 69,266      |
| SimHash 64           | 35,813        | 632,478   | 697,627     |
| SimHash 128          | 35,058        | 277,666   | 342,792     |
| CityHash 64          | 38,576        | 1,891,030 | 1,956,452   |
| CityHash 128         | 36,653        | 1,021,649 | 1,087,033   |
| CityHash 256         | 36,753        | 654,260   | 719,393     |

Table 5 shows that increasing the hash size reduces the number of false positives, thus leading to a decreased runtime. For X<sub>ASH</sub>, the number of false positives decreases when using 128 instead of 64 bits, but increases slightly with 256 bits. This is because the FP rate is already very low with 128 bits and increasing the hash size increases the runtime of the hash value checks. When using a 128-bit super key, only 5,535 FPs passed the filter compared to more than 275,000 with SimHash and more than 1,000,000 with CityHash. For SimHash and CityHash, the effect is more significant. Increasing the bits from 64 to 128 nearly halves the number of false positives for SimHash and CityHash from around 630,000 to fewer than 275,000 for SimHash and from more than 1,900,000 to around 1,000,000 for CityHash.

This experiment shows that a larger hash size generally leads to better pruning. Increasing the number of bits for the hash value reduces the number of false positives as there are more bits available to encode the row. However, as seen for `XASH`, there might be a cap on how much pruning can be achieved. It is important to note that increasing the size requires more disk space for storing the hashes. Increasing the hash size for `CityHash` from 64 bits to 256 bits quadruples the space required for storing the super keys. In particular, the super key with 64-bit hash space requires 11.3 GB and 0.4 GB for DWTC webtables and Wikipedia, respectively. This increases to 45.2 GB and 1.5 GB when using 256 bits. For large data lakes, this increase could mean significant storage space that could be saved by using a more effective hash function, such as `XASH`.

### 5.3.2 Modifying the `XASH` Generation

It is possible to modify the generation of `XASH`, either by altering the `XASH` function itself or by changing the input value based on which the hash value gets generated.

**Rotation** `XASH` uses a bit rotation routine to differentiate strings with the same rare characters but different lengths. With the rotation, it is expected that more unique hashes will be generated and as such, the number of false positives will be reduced. We test this assumption by comparing `XASH` with and without rotation on the different samples of the DWTC dataset.

Table 6 shows the number of false positives with and without rotation. Generally, the runtime differences are not statistically significant.

Tab. 6: Number of false positives with/without rotation

|               | With rotation | Without rotation |
|---------------|---------------|------------------|
| 1,000 Tables  | <b>156</b>    | 164              |
| 5,000 Tables  | 517           | <b>515</b>       |
| 10,000 Tables | <b>815</b>    | 821              |

As the shifting of the hash values consumes additional resources, the higher resource consumption makes the use of the rotation unjustifiable with the low number of additional false positives when removing the rotation.

To further explore the influence of the rotation step, we also compared the FP-rate for different table sizes. For this purpose, we sampled 383 tables with 20 columns and systematically removed columns so that the actual duplicate tables remained duplicates with just fewer columns. This experiment as well showed that turning off the rotation showed only a minor improvement in the filtering ability.

**Input String** So far, the super key for each row is generated using XASH by generating the hash value for each row value and then logically OR'ing all hash values of the row values.

A different approach for generating the super key is to concatenate all values of a row and then generate a single hash value for the concatenated string.

This would have the advantage that fewer hashes need to be generated. If the number of false positives using the concatenated input string for the hash generation is lower or equal to when OR'ing the row value hashes, the concatenation method could be preferred as the hash function will have a higher overhead.

To evaluate this theory, we obtained 1,000 tables having 20 columns from the Wikipedia dataset. The row values of each row in all tables are then sorted. After finding duplicates among the tables and recording the false positives, we remove the last column from all tables. The tables that now contain 19 columns are tested for duplicates again and the false positives are reported. This is repeated until the tables contain only 1 column each.

We ran each test twice, one time using the super keys generated by logically OR'ing the hash of the row values of each row and one time using the super key generated by concatenating the row values of each row and then generating the XASH value. The super keys were generated using XASH with 64 bits.

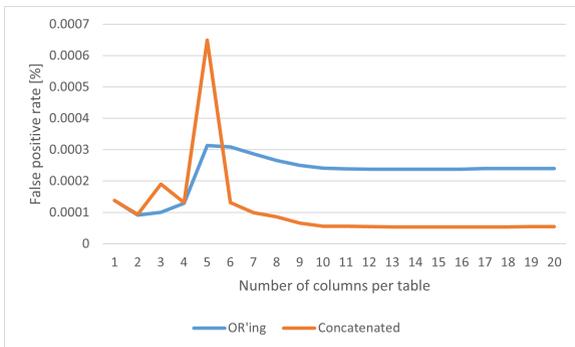


Fig. 5: False positives of XASH with different input formats

Figure 5 shows the FP rate for each group of columns per table for both approaches. It can be observed that generating the super key based on OR'ing has fewer false positives for  $<6$  columns per table, while concatenation has fewer false positives for  $\geq 6$  columns per table

The false positive rate is increased when using OR'ing when there are more columns per table. This is due to an increased number of 1-bits existing in the super key when more hash values are combined using the bitwise OR. When using concatenation to generate the super key, this problem does not occur.

This characteristic also explains the spike using concatenation on tables with fewer columns.

When using concatenation, there is still the same limited number of 1-bits is used as for more columns, while OR'ing uses more 1-bits to encode the rows more uniquely.

## 5.4 Experimental Summary

The number of false positives has a direct influence on the runtime of the pairwise table comparison and duplicate table discovery. The false positive ratio increases with the number of columns as the same hash function has to represent more information. We also conducted preliminary experiments on the effect of Null values on our hash functions. For input tables that only contain null values in one or multiple rows, all hash functions have an equally high percentage of false positives and therefore a high runtime. The same row values produce the same hash value. When the hash values are logically OR'ed, the super key generated is the same for the row, no matter whether the table consists of 1 or 10 columns.

Increasing the hash size decreased the number of false positives for all hash functions. This however leads to more storage space required to store the hashes.

## 6 Conclusion

We explored the benefits of using hash-based filters in finding duplicate tables. We showcased the duplicate table discovery use case as well as the pairwise comparison use case for lake de-duplication.

The evaluation shows that using hash functions generally improves the overall runtime. In particular, XASH shows the highest promise, followed by SimHash, CityHash, and MD5. In comparison to the original use case of multi-column join discovery, one can say that it is possible to further simplify XASH for efficiency reasons as rotation plays a minor role. The duplicate detection setting is already stricter than join discovery as the probing requires the equality of the hash functions and one-sided containment.

A challenge for all hash functions is when tables with many columns have to be encoded, as the length of a row negatively impacts the pruning power of the hash function.

Future improvements for table de-duplication could be to consider hashing for the grouping phase of large table corpora and to devise algorithms that are independent of lake indexes. Furthermore, it would be interesting to research fuzzy table duplicates. Our current approaches consider tables to be duplicates only when two tables contain the same set of tuples regardless of row and column order.

**Acknowledgements.** This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445.

## References

- [Ab16] Abedjan, Z.; Morcos, J.; Ilyas, I. F.; Ouzzani, M.; Papotti, P.; Stonebraker, M.: DataXFormer: A robust transformation discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE). IEEE Computer Society, pp. 1134–1145, 2016, URL: <https://doi.org/10.1109/ICDE.2016.7498319>.
- [Ar20] Armbrust, M.; Das, T.; Paranjpye, S.; Xin, R.; Zhu, S.; Ghodsi, A.; Yavuz, B.; Murthy, M.; Torres, J.; Sun, L.; Boncz, P. A.; Mokhtar, M.; Hovell, H. V.; Ionescu, A.; Luszczak, A.; Switakowski, M.; Ueshin, T.; Li, X.; Szafranski, M.; Senster, P.; Zaharia, M.: Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores. Proceedings of the VLDB Endowment (PVLDB)/, pp. 3411–3424, 2020.
- [Au07] Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: The Semantic Web. Springer Berlin Heidelberg, pp. 722–735, 2007.
- [Au20] Auer, S.; Oelen, A.; Haris, M.; Stocker, M.; D’Souza, J.; Farfar, K. E.; Vogt, L.; Prinz, M.; Wiens, V.; Jaradeh, M. Y. Bibliothek Forschung und Praxis 44/3, pp. 516–529, 2020, URL: <https://doi.org/10.1515/bfp-2020-2042>.
- [Br97] Broder, A. Z.; Glassman, S. C.; Manasse, M. S.; Zweig, G.: Syntactic Clustering of the Web. Comput. Networks 29/8-13, pp. 1157–1166, 1997, URL: [https://doi.org/10.1016/S0169-7552\(97\)00031-7](https://doi.org/10.1016/S0169-7552(97)00031-7).
- [CCB02] Cooper, J. W.; Coden, A.; Brown, E. W.: Detecting similar documents using salient terms. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM). ACM, pp. 245–251, 2002, URL: <https://doi.org/10.1145/584792.584835>.
- [CGS03] Conrad, J. G.; Guo, X. S.; Schriber, C. P.: Online duplicate document detection. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM). ACM Press, 2003.
- [Ch12a] Christen, P.: A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. IEEE Transactions on Knowledge and Data Engineering (TKDE) 24/9, pp. 1537–1555, 2012.
- [Ch12b] Christen, P.: Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer Publishing Company, Incorporated, 2012, ISBN: 3642311636.
- [Ch21] Christophides, V.; Efthymiou, V.; Palpanas, T.; Papadakis, G.; Stefanidis, K.: An Overview of End-to-End Entity Resolution for Big Data. ACM Comput. Surv. 53/6, 127:1–127:42, 2021, URL: <https://doi.org/10.1145/3418896>.
- [Eb15a] Eberius, J.: The Dresden Web Table Corpus, 2015, URL: <https://wwwdb.inf.tu-dresden.de/misc/dwtc/>, visited on: 04/27/2022.

- [Eb15b] Eberius, J.; Thiele, M.; Braunschweig, K.; Lehner, W.: Top-k Entity Augmentation Using Consistent Set Covering. In: Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM). Association for Computing Machinery, 2015.
- [EQA22] Esmailoghli, M.; Quiané-Ruiz, J.-A.; Abedjan, Z.: MATE: Multi-Attribute Table Extraction. In: Proceedings of the VLDB Endowment (PVLDB). Apr. 2022.
- [Fe18] Fernandez, R. C.; Abedjan, Z.; Koko, F.; Yuan, G.; Madden, S.; Stonebraker, M.: Aurum: A Data Discovery System. In: Proceedings of the International Conference on Data Engineering (ICDE). IEEE Computer Society, pp. 1001–1012, 2018, URL: <https://doi.org/10.1109/ICDE.2018.00094>.
- [Fi15] Fisher, J.; Christen, P.; Wang, Q.; Rahm, E.: A Clustering-Based Framework to Control Block Sizes for Entity Resolution. In: KDD '15, Association for Computing Machinery, Sydney, NSW, Australia, pp. 279–288, 2015, ISBN: 9781450336642, URL: <https://doi.org/10.1145/2783258.2783396>.
- [FSF20] Fernandez, R. C.; Subramaniam, P.; Franklin, M. J.: Data Market Platforms: Trading Data Assets to Solve Data Problems. Proceedings of the VLDB Endowment (PVLDB), pp. 1933–1947, 2020.
- [Ga22] Gagliardelli, L.; Papadakis, G.; Simonini, G.; Bergamaschi, S.; Palpanas, T.: Generalized Supervised Meta-blocking. Proceedings of the VLDB Endowment (PVLDB) 15/9, pp. 1902–1910, 2022, URL: <https://www.vldb.org/pvldb/vol15/p1902-gagliardelli.pdf>.
- [GF98] Grossman, D. A.; Frieder, O. In: Information Retrieval: Algorithms and Heuristics. Springer US, pp. 134–137, 1998.
- [He06] Henzinger, M. R.: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR). ACM, pp. 284–291, 2006, URL: <https://doi.org/10.1145/1148170.1148222>.
- [Ja19] Jaradeh, M. Y.; Oelen, A.; Farfar, K. E.; Prinz, M.; D'Souza, J.; Kismihók, G.; Stocker, M.; Auer, S.: Open Research Knowledge Graph: Next Generation Infrastructure for Semantic Scholarly Knowledge. In: Proceedings of the 10th International Conference on Knowledge Capture. K-CAP '19, Association for Computing Machinery, Marina Del Rey, CA, USA, pp. 243–246, 2019, ISBN: 9781450370080, URL: <https://doi.org/10.1145/3360901.3364435>.
- [Jo72] Jones, K. S.: A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28/1, pp. 11–21, Jan. 1972.
- [KPN20] Koumarelas, I. K.; Papenbrock, T.; Naumann, F.: MDedup: Duplicate Detection with Matching Dependencies. Proceedings of the VLDB Endowment (PVLDB) 13/5, pp. 712–725, 2020, URL: <http://www.vldb.org/pvldb/vol13/p712-koumarelas.pdf>.

- [KTR10] Köpcke, H.; Thor, A.; Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. Proceedings of the VLDB Endowment (PVLDB) 3/1, pp. 484–493, 2010, URL: [http://www.vldb.org/pvldb/vldb2010/pvldb%5C\\_vol13/E04.pdf](http://www.vldb.org/pvldb/vldb2010/pvldb%5C_vol13/E04.pdf).
- [Ku17] Kumar, N.; Antwal, S.; Samarthyam, G.; Jain, S.: Genetic optimized data deduplication for distributed big data storage systems. In: 2017 4th International Conference on Signal Processing, Computing and Control (ISPC). Sept. 2017.
- [Li20] Li, Y.; Li, J.; Suhara, Y.; Doan, A.; Tan, W.: Deep Entity Matching with Pre-Trained Language Models. Proceedings of the VLDB Endowment (PVLDB) 14/1, pp. 50–60, 2020, URL: <http://www.vldb.org/pvldb/vol14/p50-li.pdf>.
- [LSR21] Lerm, S.; Saeedi, A.; Rahm, E.: Extended Affinity Propagation Clustering for Multi-source Entity Resolution. In: Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“(DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings. Vol. P-311. LNI, Gesellschaft für Informatik, Bonn, pp. 217–236, 2021, URL: <https://doi.org/10.18420/btw2021-11>.
- [MJS07] Manku, G. S.; Jain, A.; Sarma, A. D.: Detecting near-duplicates for web crawling. In: Proceedings of the International World Wide Web Conference (WWW). ACM Press, 2007.
- [Na18] Nargesian, F.; Zhu, E.; Pu, K. Q.; Miller, R. J.: Table Union Search on Open Data. Proceedings of the VLDB Endowment (PVLDB) 11/7, pp. 813–825, 2018, URL: <http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf>.
- [PA22] Pike, G.; Alakuijala, J.: Introducing CityHash | Google Open Source Blog, 2022, URL: <https://opensource.googleblog.com/2011/04/introducing-cityhash.html>, visited on: 08/25/2022.
- [Si22] Simonini, G.; Zecchini, L.; Bergamaschi, S.; Naumann, F.: Entity Resolution On-Demand. Proceedings of the VLDB Endowment (PVLDB) 15/7, pp. 1506–1518, 2022, URL: <https://www.vldb.org/pvldb/vol15/p1506-simonini.pdf>.
- [Te22] Technische Informationsbibliothek: Comparisons - ORKG, 2022, URL: <https://orkg.org/about/15/Comparisons>, visited on: 08/02/2022.
- [Th20] Thirumuruganathan, S.; Tang, N.; Ouzzani, M.; Doan, A.: Data Curation with Deep Learning. In: Proceedings of the International Conference on Extending Database Technology (EDBT). OpenProceedings.org, pp. 277–286, 2020, URL: <https://doi.org/10.5441/002/edbt.2020.25>.
- [TSP08] Theobald, M.; Siddharth, J.; Paepcke, A.: SpotSigs: robust and efficient near duplicate detection in large web collections. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR). ACM, pp. 563–570, 2008, URL: <https://doi.org/10.1145/1390334.1390431>.

- [WLF11] Wang, J.; Li, G.; Feng, J.: Fast-join: An efficient method for fuzzy token matching based string similarity join. In: Proceedings of the International Conference on Data Engineering (ICDE). IEEE Computer Society, pp. 458–469, 2011, URL: <https://doi.org/10.1109/ICDE.2011.5767865>.
- [Wo22] World Bank: GDP (current US\$), Apr. 2022, URL: <https://data.worldbank.org/indicator/NY.GDP.MKTP.CD>, visited on: 04/27/2022.
- [Xi11] Xiao, C.; Wang, W.; Lin, X.; Yu, J. X.; Wang, G.: Efficient similarity joins for near-duplicate detection. ACM Transactions on Database Systems (TODS) 36/3, pp. 1–41, 2011.
- [Yu16] Yu, M.; Li, G.; Deng, D.; Feng, J.: String similarity search and join: a survey. Frontiers Comput. Sci. 10/3, pp. 399–417, 2016, URL: <https://doi.org/10.1007/s11704-015-5900-5>.
- [Zh19] Zhu, E.; Deng, D.; Nargesian, F.; Miller, R. J.: JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In: Proceedings of the International Conference on Management of Data (SIGMOD). ACM, pp. 847–864, 2019, URL: <https://doi.org/10.1145/3299869.3300065>.

# DPQL: The Data Profiling Query Language

Marcian Seeger<sup>1</sup>, Sebastian Schmidl<sup>2</sup>, Alexander Vielhauer<sup>3</sup>, Thorsten Papenbrock<sup>4</sup>

**Abstract:** Data profiling describes the activity of extracting implicit metadata, such as schema descriptions, data types, and various kinds of data dependencies, from a given data set. The considerable amount of research papers about novel metadata types and ever-faster data profiling algorithms emphasize the importance of data profiling in practice. Unfortunately, though, the current state of data profiling research fails to address practical application needs: Typical data profiling algorithms (i. e., challenging to operate structures) discover all (i. e., too many) minimal (i. e., the wrong) data dependencies within minutes to hours (i. e., too long). Consequently, if we look at the practical success of our research, we find that data profiling targets data cleaning, but most cleaning systems still use only hand-picked dependencies; data profiling targets query optimization, but hardly any query optimizer uses modern discovery algorithms for dependency extraction; data profiling targets data integration, but the application of automatically discovered dependencies for matching purposes is yet to be shown - and the list goes on. We aim to solve the profiling-and-application-disconnect with a novel data profiling engine that integrates modern profiling techniques for various types of data dependencies and provides the applications with a versatile, intuitive, and declarative Data Profiling Query Language (DPQL). The DPQL enables applications to specify precisely what dependencies are needed, which not only refines the results and makes the data profiling process more accessible but also enables much faster and (in terms of dependency types and selections) holistic profiling runs. We expect that integrating modern data profiling techniques and the post-processing of their results under a single application endpoint will result in a series of significant algorithmic advances, new pruning concepts, and a profiling engine with innovative components for workload autoconfiguration, query optimization, and parallelization. With this paper, we present the first version of the DPQL syntax and its semantics, which introduces a fundamentally new line of research in data profiling.

**Keywords:** data profiling; query language; functional dependencies; unique column combinations; inclusion dependencies

## 1 About Data Profiling and Application Requirements

Structural metadata is a set of rules that shape datasets, their formats, evolution, correctness, and accessibility. For this reason, metadata is an essential input to many data management processes ranging from data exploration [Fe18; Ro09] over data integration [DR02; Zh10]

---

<sup>1</sup> Philipps-University of Marburg, Big Data Analytics, Hans-Meerwein-Str. 6, 35032 Marburg, Germany  
seegerma@students.uni-marburg.de

<sup>2</sup> Hasso Plattner Institute, University of Potsdam, Information Systems, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany  
sebastian.schmidl@hpi.de

<sup>3</sup> Philipps-University of Marburg, Big Data Analytics, Hans-Meerwein-Str. 6, 35032 Marburg, Germany  
avielhauer@informatik.uni-marburg.de

<sup>4</sup> Philipps-University of Marburg, Big Data Analytics, Hans-Meerwein-Str. 6, 35032 Marburg, Germany  
papenbrock@informatik.uni-marburg.de

and data cleaning [IC15; VA18] to query optimization [KPN22; Pa00] and machine learning [Ch19; KPN20] to name only a few. Due to the importance of metadata, most database management systems store not only the data but also structural information, such as data types, basic statistics, and constraints. This is only a fraction of the structural metadata that characterizes a dataset, and actually having access to it is a lucky case because many formats and systems for storing datasets do not even provide any metadata. For this reason, data engineers (and scientists) conduct *data profiling* [Ab18] to extract metadata from raw data. This first manual and meanwhile largely automated process has been improved significantly over the past 30 years. To give a few examples, we can now automatically mine *unique column combinations* [Bi20], *functional dependencies* [PN16], *inclusion dependencies* [Dü19], *order dependencies* [SP22], *matching dependencies* [Sc20] and many more in exact and various relaxed versions [CDP16]. In the quest to meet application needs and user skills, the corresponding data profiling algorithms have been built into practical data profiling tools, such as Metanome [Pa15a], Desbordante [De22], or Viadotto [Vi22]. Despite these technological advances, data profiling still requires complicated and often manual post-processing efforts to make use of the discovered metadata in different applications.

To illustrate the current limitations in data profiling, consider the following example: In a data integration scenario, a data engineer is looking for possible foreign-key candidates between two to-be-integrated datasets  $R$  and  $S$ . A suitable foreign-key candidate is an inclusion dependency (IND)  $X \subseteq Y$  where the attributes  $X$  and  $Y$  are from different datasets and the target  $Y$  is a key candidate, i. e., a unique column combination (UCC). The standard approach would be to, first, discover all INDs and UCCs and, then, filter the required statements for the actual results. This process introduces the following major issues:

**Discovery of too many results:** Many data profiling algorithms are exponential in their output complexity because the amount of syntactically valid and, hence, discoverable metadata is huge. They usually restrict the outputs to only minimal (or maximal) metadata statements, but the metadata result sets still often outgrow storage capacities and the data itself [Dü19; Pa15b]. Any semantic metadata selection is usually conducted as a post-processing step and deferred to the metadata application. If these applications could formulate their metadata requirements as pruning rules for the profiling of the data, giant result sets could be avoided. In our example, only very few INDs overlap with a UCC, such that a clever profiling run would never need to enumerate all INDs and UCCs.

**Discovery of the wrong results:** To limit the size of metadata result sets, the profiling algorithms restrict the enumeration to only minimal (or maximal) statements. It is possible to derive any valid metadata statement from these collections, but the inference requires complex post-processing procedures based on different axiomatizations [Ab18]. If the INDs in our example are all maximal and the UCCs are all minimal, then the needed foreign-key candidates might be formed by an IND-UCC-combination in which neither the IND is maximal nor the UCC is minimal; and apart from linking the two dependency statements, additional inference work based on IND- and UCC-axioms is needed. A sophisticated

profiling algorithm would do this already during metadata discovery, which requires a standardized profiling language to configure the algorithm executions accordingly.

**Discovery in the wrong way:** All state-of-the-art data profiling tools operate on a one-type-at-a-time basis, which means that they offer distinct discovery functionalities for every type of metadata. To fulfill a certain metadata need, a user must first decompose the application demands into these different metadata types. For each type, the best algorithm then needs to be selected and parameterized. The latter involves specifying whether relaxation, approximation, parallelization, disk-swapping, etc. is needed and if yes, to what degree. The user must also configure algorithm-specific parameters, such as window sizes, search depths, or filter sizes. This complexity prevents many users from applying modern data profiling tools. For the foreign-key discovery example, the data engineer needs to parameterize an IND and UCC algorithm and combine the results, which is something that a holistic, application-driven data profiling tool with a simple, declarative query language should be able to do automatically.

**Discovery that takes too long:** Data profiling algorithms are highly optimized, extremely effective metadata discovery tools; they are still output bound, and the outputs grow exponentially with the input sizes. For this reason, even the most effective algorithms may take hours to days to enumerate complete metadata sets for certain inputs [Kr16]. The only way to achieve further significant performance improvements is to pull the application-specific selection of the metadata statements from the preprocessing into the profiling algorithms. This requires a generic language for pruning rules and holistic profiling algorithms that discover multiple types of metadata simultaneously. In our example, we aim to discover INDs and UCCs simultaneously and, for this, need to specify the relationship between them. These relationships can be specified with a data profiling query language and translate directly into pruning rules for the discovery.

A holistic data profiling engine with a standardized *Data Profiling Query Language (DPQL)* would resolve all four mentioned issues: The declarative query language serves to formulate exactly what metadata statements are needed, such that only truly required results (not too many) and carefully linked results (not the wrong) are discovered. Based on the explicit metadata queries, the data profiling engine can automate the parameterization (not in the wrong way) and optimize the discovery strategy (not too long). In this paper, we introduce the first version of such a data profiling query language and provide concrete examples of its usage. DPQL is a generic, SQL-like language that serves to specify metadata requirements across different metadata types. From a user perspective, DPQL is an intuitive interface to filter and join metadata statements that are transparently discovered on demand.

Holistic data profiling via a standardized, declarative metadata query language is a fundamentally new approach to data profiling and should have a major impact on how data profiling algorithms and tools are developed in the future. The descriptions of the DPQL language in this paper focus on the three most popular data dependencies, which are UCCs, FDs, and INDs, but they generalize to all other types of dependencies and metadata statements.

In the subsequent sections, we first introduce a small running example with a practical DPQL query (Sect. 2). Then, we discuss related work on data profiling and profiling-related query languages (Sect. 3) and recap the definitions of UCCs, FDs, and INDs (Sect. 4). Afterward, we introduce DPQL’s **SELECT-FROM-WHERE** syntax and result format (Sect. 5). We then present the novel functions that can be used within a DPQL query and explain their purposes (Sect. 6). As an evaluation of DPQL, we consider different application areas of data profiling and formulate their demands in DPQL (Sect. 7). In the end, we motivate novel research challenges inspired by DPQL (Sect. 8) and summarize our proposal (Sect. 9).

## 2 A Running Example

| ID | Name    | Evolution | Location        | Sex | Weight | Size | Type     | Weak   | Strong   |
|----|---------|-----------|-----------------|-----|--------|------|----------|--------|----------|
| 25 | Pikachu | Raichu    | Viridian Forest | m/f | 6.0    | 0.4  | electric | ground | water    |
| 29 | Nidoran | Nidorino  | Safari Zone     | m   | 9.0    | 0.5  | poison   | ground | gras     |
| 32 | Nidoran | Nidorina  | Safari Zone     | f   | 7.0    | 0.4  | poison   | ground | gras     |
| 63 | Abra    | Kadabra   | Cerulean Cave   | m/f | 19.5   | 0.9  | psychic  | ghost  | fighting |
| 64 | Kadabra | Simsala   | Cerulean Cave   | m/f | 56.5   | 1.3  | psychic  | ghost  | fighting |

(a) Pokemon

| Title           | Biome    | Region |
|-----------------|----------|--------|
| Viridian Forest | gras     | Kanto  |
| Safari Zone     | gras     | Kanto  |
| Cerulean Cave   | rock     | Kanto  |
| Fuchsia City    | fighting | Kanto  |
| Anemonia City   | water    | Jotho  |

(b) Locations

| Firstname | Rank | Pokecount |
|-----------|------|-----------|
| Marcian   | 8    | 38        |
| Sebastian | 5    | 42        |
| Alexander | 2    | 19        |
| Thorsten  | 1    | 2         |
| Elisa     | 9    | 73        |

(c) Trainers

| Trainer   | Pokemon |
|-----------|---------|
| Marcian   | 64      |
| Elisa     | 29      |
| Elisa     | 32      |
| Sebastian | 25      |
| Sebastian | 64      |

(d) Teams

Tab. 1: A running example with a small excerpt of Pokémon data.

As an introduction to DPQL, let us examine a small example with the Pokémon data shown in Tab. 1. In this example, we aim to discover all foreign-key relationships between the tables in the Pokémon dataset. A foreign-key is an integrity constraint between two lists of attributes that requires an *inclusion dependency* between the attribute lists and a *unique column combination* on the referenced attribute list. In some domain-specific settings, we might also want these relationships to cover at most *two attributes*, to link attributes from *different tables*, and to have at

```

1  SELECT
2  X AS ForeignKey, Y AS Key
3  FROM
4  CC(Pokemon,Locations,Trainers,Teams) X,
5  CC(Pokemon,Locations,Trainers,Teams) Y
6  WHERE
7  IND(X,Y)
8  AND UCC(Y)
9  AND SPLIT(X,Y)
10 AND SIZE(Y) <= 2
11 AND CARDINALITY(X) >= 2

```

List. 1: Find all foreign-key candidates between the tables *Pokemon*, *Locations*, *Trainers*, and *Teams*.

*least two different values* in the foreign-key columns. All these conditions can be formulated in a single DPQL query, such as the one shown in List. 1. Following the SQL syntax, the **SELECT** clause defines the output to be pairs of attribute lists  $X$  and  $Y$ , which represent the needed **ForeignKey** and **Key** attributes, respectively. The **FROM** clause specifies the search space for  $X$  and  $Y$  with the help of the **CC()** function that describes all possible *column combinations* of its arguments; for this, DPQL needs to be able to fetch attribute information from relations. The **WHERE** clause specifies the constraints on the metadata that we are looking for: the **IND**  $X \subseteq Y$  and the **UCC**  $Y$  should be valid (**IND**( $X, Y$ ) and **UCC**( $Y$ )),  $X$  and  $Y$  should be from different relations (**SPLIT**( $X, Y$ )), the size of the foreign-key should not be greater than two (**SIZE**( $Y$ )  $\leq 2$ ), and  $X$  should contain at least two different values (**CARDINALITY**( $X$ )  $\geq 2$ ). The answer to this query contains the tuples ( $[Trainer]$ ,  $[Firstname]$ ), ( $[Pokemon]$ ,  $[ID]$ ), and ( $[Location]$ ,  $[Title]$ ), which are precisely the foreign-keys of the Pokémon dataset.

To obtain the result of a DPQL query, a novel data profiling engine is needed that can parse the filter criteria from the query and apply them effectively. Note that the query implies many implicit profiling constraints, e. g., that  $X$  and  $Y$  need to be of the same size, both column combinations need to be lists while column combinations that do not appear in **INDs** can be interpreted as sets, and the results should be minimal/maximal according to dependency axioms. These constraints do not need to be specified and can automatically be derived by the profiling engine and algorithms. To the best of our knowledge, not a single existing data profiling system can consider all such profiling constraints.

For demonstration purposes, we implemented a very early query processing prototype for the DPQL language that can answer the queries shown in this paper. With the prototype, we executed the foreign-key query of List. 1 on the TPC-H (425 MB, 7 Tables, and 56 Attributes) and the MusicBrainz (104 GB, 232 Tables, and 1 562 Attributes) datasets: The foreign-key query on the TPC-H dataset yields 19 foreign-key candidates containing all seven true foreign-key constraints; in contrast, a full profiling run yields 52 maximal **INDs** and 408 minimal **UCCs** that still need to be combined. The foreign-key query on the MusicBrainz dataset yields 7 625 foreign-key candidates; in contrast, a full profiling run yields 209 572 unary **INDs** and 496 minimal **UCCs** that still need to be combined. These experiments demonstrate that DPQL queries can produce smaller and more specific results; it enables holistic profiling and new pruning rules for faster executions.

We need to emphasize that most real-world datasets are much wider and longer than our tiny Pokémon example dataset; additionally, they often lack descriptive labels, offer only cryptic values, and are hard to parse. For this reason, automatic data profiling starting at the source data and delivering suitable results directly to the applications – just as we did with the foreign-key query – is highly needed. DPQL is a first and essential building block for this.

### 3 Related Work

Most of the research on data profiling focuses primarily on improving existing data profiling algorithms including aspects, such as their scalability [Sc20; SGI19; SP22], relaxation [Ca21b; CDP16; Li20; WHL21], or dynamics [Ca21a; Xi22]. Consequently, many very effective algorithms exist for the discovery of basic metadata [HN17; HPN21], unique column combinations [Bi20; WLL19], functional dependencies [PN16; WLL19], inclusion dependencies [Dü19; Pa15c], order dependencies [SP22; Sz17], matching dependencies [Sc20; Wa17], denial constraints [BKN17; PAN21] and many further. All these algorithms target only one type of dependency and try to enumerate complete result sets; they are written in different languages and serve quite heterogeneous interfaces and result formats, which makes them relatively difficult to apply in real-world settings.

So far, very little research has been done on holistic profiling techniques. Some works exist that consider UCCs and FDs simultaneously [Eh16; Hu99] or reason about FDs and INDs jointly [HL18]. These approaches demonstrate the potential of holistic data profiling w.r.t. runtime improvements, but a query language is needed to cover more than two types of dependencies and semantically combine and filter the results.

Data profiling tools aggregate profiling algorithms and present them as services to the user. They make the algorithms easier to operate and store the results in some tool-specific but at least type-unified format. The open-source research framework *Metanome* [Pa15a] was the first data profiling tool to support the discovery of various types of metadata. Another very recent profiling tool inspired by Metanome is *Desbordante* [De22]. Meanwhile, commercial products, such as *Viadotto* [Vi22], developed the idea further and professionalized the concepts. All these tools effectively ease the profiling for non-expert users, but since they do not offer any metadata management features, they effectively shifted the problem from complicated-to-operate data to complicated-to-operate metadata.

A promising approach to the metadata management concern is to store the discovered metadata in the form of data profiles in a database. In this way, users can issue SQL queries to find, join, and filter the metadata according to their specific application needs. A practical implementation of this idea, which works nicely with Metanome, is the metadata management system *Metacrate* [Kr17a]. Metacrate proposes effective, relational storage formats for various types of metadata, and SQL as a flexible and generic query language. Another metadata store that focuses on statistical metadata rather than structural metadata is *Splash* [FL10]. Similar to Metacrate, Splash is based on SQL and tries to persist all metadata. The general approach of persisting the metadata, though, comes with various issues: Synchronization of data and metadata is expensive, metadata contains a lot of redundancy, schemata with many types (UCCs, INDs, FDs, ODs, . . .) and relaxations (partial, approximate, conditional, . . .) become incomprehensible, and, most importantly, metadata sets are huge if they are stored in their entirety. Standard SQL also appears to be an unfavorable match for working with metadata, which is why we propose a novel query language and a profiling engine that collects the metadata at query time.

As part of our related work, we also consider the enormous space of business intelligence systems with data profiling capabilities, including IBM InfoSphere, Talent Data Quality, Informatica Data Explorer, Trillium Software Data Profiling, OpenRefine, SAP Business Objects, and many, many more. Apart from the fact that many data profiling features of these tools are still behind state-of-the-art in research, they suffer from the same metadata management and accessibility issues as the whole field of data profiling.

Because metadata statements, and data dependencies in particular, are defined on schema level, a data profiling query language needs to be able to access schema elements. *SchemaSQL* [LSS96] is an SQL extension that already offers these capabilities: The queries can access database-, relation-, and attribute-names, join values with labels, compare schema elements, and alter them. The only schema operation needed for data profiling, though, is referring to attribute lists. For this reason and because SchemaSQL also lacks data profiling features, we create a new SQL-like dialect.

Defining SQL extensions or entirely new query languages to query derived information is not a novelty. In the data mining area, which is closely related to data profiling, various efforts have been made to extend SQL with data mining capabilities. For example, *MINE RULE* is a keyword extension to discover rules [MPC+96; MPC98] and the *profile* function is Splash's extension to extract estimated joint probability density functions [FL10]. Data mining algorithms have also been defined via user-defined functions [OP11] or virtual views [B112]. Similar extensions would be possible also for data profiling algorithms, but a query language specifically designed for data profiling is easier to understand, clearer in semantics and result formats, and better to be parsed into data profiling pruning rules.

The SQL-like data mining language *RQL* [Ch17] is a query language for discovering exact, extended and relaxed functional dependencies. It is the closest challenger of our proposal, but it can discover only simple *if-then*-statements and no arbitrary complex metadata constructs with different types of metadata. The extension of RQL to a more comprehensive data profiling language would change not only the language, but also its execution engine significantly. Therefore, we propose a novel, more intuitive query language.

## 4 Data Dependencies

Throughout the paper, we follow established notations for data profiling [Ab18]: Because these notations consider schemata and data to be ordered (e. g. by their physical order on disk), we use the terms *attribute* and *column*, as well as *record*, *tuple*, and *row* interchangeably. We denote a relational schema as  $R$  and instances of  $R$  as  $r$ . Letters from the start of the alphabet denote attributes ( $A, B, C, D, \dots \in R$ ) and letters from the end of the alphabet denote attribute lists ( $\dots, W, X, Y, Z \subseteq R$ ). Attributes in these lists can be accessed via index, e. g., as  $X_i$ . For some metadata statements, the order of the attributes in attribute lists is important (e. g. INDs) and for others it is not (e. g. UCCs and FDs). We, therefore, name these lists *column combinations* and let the profiling algorithm infer, based on the type of

dependency, whether a combination needs to be interpreted as a list or a set. The notations  $R[X]$  and  $t[X]$  denote projections of schema  $R$  and tuple  $t$  on the attributes  $X$ , respectively. With these notations, we define UCCs, FDs, and INDs as follows:

**Definition 1 (Unique column combination (UCC))** *Given a schema  $R$  with instance  $r$ , a UCC  $X$  with  $X \subseteq R$  is valid in  $r$ , iff  $\forall t_i, t_j \in r, i \neq j : t_i[X] \neq t_j[X]$ .*

**Definition 2 (Functional dependency (FD))** *Given a schema  $R$  with instance  $r$ , the FD  $X \rightarrow A$  with  $X \subseteq R$  and  $A \in R$  is valid in  $r$  iff  $\forall t_i, t_j \in r : t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A]$ .*

**Definition 3 (Inclusion dependency (IND))** *Given the schemata  $R$  and  $S$  with instances  $r$  and  $s$ , respectively, the IND  $R[X] \subseteq S[Y]$  (abbreviated  $X \subseteq Y$ ) with attribute lists  $X \subseteq R$  and  $Y \subseteq S$ , and cardinalities  $|X| = |Y|$  is valid iff  $\forall t_i \in r, \exists t_j \in s : t_i[X] = t_j[Y]$ .*

Considering our running example in Tab. 1, we find that, for example,  $\{Name, Sex\}$  is a UCC,  $\{Type\} \rightarrow \{Weak\}$  is an FD, and  $\{Location\} \subseteq \{Title\}$  is an IND. Because UCCs indicate keys, FDs indicate value associations, and INDs indicate referential integrity, these three dependencies are among the most important metadata statements. For more details on axiomatization, inference rules, and minimality/maximality properties, we refer to [Ab18]. In the context of this paper, it should be sufficient to understand that all profiling-related aspects are pushed down to the profiling engine and/or algorithm(s).

## 5 Data Profiling Query Language

The *Data Profiling Query Language* (DPQL) is a variant of SQL that follows the popular **SELECT-FROM-WHERE** syntax. A central element of this syntax is the *column combination* function **CC()**. This function allows DPQL to access schema elements as values. In the following, we first introduce the **CC()** function and, then, discuss the DPQL query syntax.

### 5.1 DPQL Column Combination Function

Data profiling is about discovering metadata statements on column combinations. With the *column combination* function **CC()**, the user can refer to these groups of attributes and, then, specify restrictions and connections for them. The parameter list of the **CC()** function is a list of relational attributes from which the column combinations should be drawn. For example, **CC(Pokemon.ID, Pokemon.Size)** describes the following list-based column combinations:  $\{\emptyset, [Pokemon.ID], [Pokemon.Size], [Pokemon.ID, Pokemon.Size], [Pokemon.Size, Pokemon.ID]\}$ . We can enumerate these from the list of attributes when considering the power set lattice of these attributes [Ab18]. While the **CC()** function defines

the *origin* of the columns, we later introduce further restrictions on column combinations that filter concrete patterns of specific dependency types. Note that the `CC()` function in DPQL is used as a *declarative* construct to define sets of column combinations, which can be named in the queries. `CC()` provides the context for the data profiling and describes the search space for the profiling algorithms, but it is not supposed to be fully materialized.

The attributes for a `CC()` call can be provided explicitly, collectively via their relations, or as negations; we can also specify concrete column combination sets as literals. Tab. 2 provides an overview of the specification options for column combinations:

**Attributes:** The most basic call of the `CC()` function lists all relational *attributes* that should be considered for the generation of column combinations. If the parameter contains attributes from different tables, these attributes will also form column combinations. For many types of metadata, such as FDs, UCCs, and INs, all attributes of a column combination must stem from the same relation and the profiling algorithms will prune the search space accordingly; for some types, such as MDs and DCs, mixed column combinations are needed, though.

**Relations:** By specifying *relations* in the `CC()` parameter lists, we denote all attributes of the respective relations. This shortcut is well established in the data profiling community, as most data profiling algorithms operate on this abstraction level.

**Negations:** With *negations*, the user can exclude certain attributes from relations in a `CC()` call. In Tab. 2, we consider all attributes in the `Pokemon` relation and exclude only the `Pokemon.ID` attribute from it. The negation is particularly useful to profile the majority of attributes while excluding certain irrelevant attributes, such as empty, generated, or binary attributes. Note that negative statements supersede positive statements, and a negative statement without a positive relation is redundant.

**Literal:** Instead of modeling the search space with the `CC()` function, sets of column combinations can also be specified explicitly with a *literal* statement. A literal groups one or multiple column combinations, which are represented as attributes in square brackets, into a set in curly brackets. A literal takes the place of any `CC()` call and can contain arbitrary many column combinations, which the profiling uses exactly as specified. If the literal cannot be parsed into a valid column combination, an error is thrown. The option to provide fixed column combinations is important to ask specific profiling questions, such as "*Where does this foreign-key point to?*" or "*Which attributes functionally depend on this key candidate?*".

| Parameter  | Example  | Description: All CCs formable with . . .      |
|------------|--|---|
| Attributes | <code>CC(Pokemon.ID, Pokemon.Size)</code>                    | the provided attributes.                      |
| Relations  | <code>CC(Pokemon, Teams)</code>                              | the attributes of the provided relations.     |
| Negations  | <code>CC(Pokemon, !Pokemon.ID)</code>                        | all attributes but the provided exceptions.   |
| Literals   | <code>{ [Pokemon.ID, Pokemon.Size], [Trainers.Rank] }</code> | exactly the two provided column combinations. |

Tab. 2: Specification options for column combination sets.

Because the order of column combinations matters for INDs (and some other dependencies), column combinations in DPQL queries are always considered as lists, i. e., the order of attributes in column combinations is meaningful. However, for set-based data dependencies, such as UCCs and FDs, the profiling engine automatically prunes redundant results.

## 5.2 DPQL Query Syntax

We now introduce the **SELECT-FROM-WHERE** syntax of DPQL and provide further examples of DPQL queries. The queries use the **CC()** function to reference column combinations (short CCs) and the functions **UCC()**, **FD()**, and **IND()** to specify dependencies (and their interactions); we provide more details on the metadata discovery functions later in Sect. 6.

**SELECT** The **SELECT** clause defines the column combinations that shall appear in the query's result. Each listed column combination translates into a column in the relational output, and every row in the relational output is a set of column combinations that answers the DPQL query. Column combinations refer to the search spaces defined by the **CC()** calls in the **FROM** clause, and can be renamed with the **AS** keyword.

```

1  SELECT
2    X AS Left, Y AS Right
3  FROM
4    CC(Pokemon,Trainers) X,
5    CC(Pokemon,Trainers) Y
6  WHERE
7    FD(X,Y)

```

List. 2: Find all functional dependencies in the relations *Pokemon* and *Trainers*.

List. 2 shows a DPQL query with a simple

**SELECT** clause that selects the left- and right-hand-sides of functional dependencies within the relations *Pokemon* and *Trainers*. Result tuples of this query would be (*[Pokemon.Type]*, *[Pokemon.Weak]*) or (*[Trainer.Rank]*, *[Trainer.Pokecount]*), which represent the FDs  $Type \rightarrow Weak$  and  $Rank \rightarrow Pokecount$ . Note that **SELECT** describes a projection on the column combinations defined in the **FROM** clause and, therefore, does not need to list all CCs – if we require only left-hand-sides, we would project on *X* alone in List. 2.

**FROM** The **FROM** clause uses the **CC()** function (or literals) to specify the search space of the profiling. Every **CC()**-defined set of column combinations needs to be named, such that it can be referenced in the **SELECT** and/or **WHERE** clause. The DPQL query in List. 3 demonstrates the discovery of inclusion dependencies in the Pokémon example with two different **CC()** sets. The results of this query contain all *X* and *Y* column combination pairs, for which the *X* values link *Pokemon* to *Y* values in either *Locations* or *Teams*.

```

1  SELECT
2    X AS Dependent, Y AS Referenced
3  FROM
4    CC(Pokemon) X,
5    CC(Locations,Teams) Y
6  WHERE
7    IND(X,Y)

```

List. 3: Find all inclusion dependencies from the relation *Pokemon* to either *Locations* or *Teams*.

**WHERE** The **WHERE** clause is a logical filter expression. It defines the metadata patterns that serve a specific application need and follows SQL operator precedence. While the **FROM** clause restricts the data profiling process to certain tables (and attributes), the **WHERE** clause can be used to formulate conditions and metadata patterns that further prune the metadata search space. Handed over to the actual data profiling, these restrictions can greatly reduce runtime and memory consumption. Expressions in the **WHERE** clause are based on data profiling functions that cover different types of metadata statements, such as **UCC()**, **FD()**, and **IND()**, and additional restrictions on column combinations, such as **SIZE()**, **MIN()**, and **MAX()** (more details in Sect. 6). Filter criteria can be linked via **AND** and **OR**, and any valid answer to a DPQL query needs to fulfill the entire **WHERE** clause. An example with a slightly larger **WHERE** clause than before is shown in List. 4: The query asks for all inclusion and functional dependencies that point to unique column combinations. Both (*Pokemon.Name*, *Pokemon.Sex*], [*Pokemon.ID*]) and (*Teams.Pokemon*], [*Pokemon.ID*]) are valid answers to the query, the former being the FD  $\{Name, Sex\} \rightarrow \{ID\}$  and the latter the IND  $\{Pokemon\} \subseteq \{ID\}$ ; the result does not differentiate FDs and INDs, but the way this query is issued (via **OR**) indicates that this information is irrelevant for the application.

```

1  SELECT
2    X Determinant, Y AS Unique
3  FROM
4    CC(Teams, Trainers) X,
5    CC(Pokemon) Y
6  WHERE
7    UCC(Y)
8    AND (IND(X, Y) OR FD(X, Y))

```

List. 4: Find all unique column combinations that are a target of a functional or inclusion dependency.

### 5.3 DPQL Result Format

In contrast to SQL, which queries database records, DPQL extracts statements about the schemata, i. e., combinations of attributes and their interactions. These schema statements are compositions of column combinations, which introduce special challenges for the output format. To understand these challenges and our design decisions for overcoming them, we first describe the straightforward case and address the complicated situations afterwards.

#### Basic DPQL Results

A DPQL query returns a result in relational format: The **SELECT** clause determines the schema of the result table by turning every provided column combination variable, which is a **CC()** call, into a relational attribute. The name of each result attribute is equal to the column combination's variable name or, if provided, its **AS**-alias. Each row in the result relation is a valid response to the DPQL query; structurally, a response row is a set of column combinations, which is a set of attribute lists. For example, Tab. 3 lists the results of

our introductory foreign-key example (see List. 1). Each of the three result tuples describes a valid foreign-key candidate according to the specified filter criteria.

| ForeignKey          | Key             |
|---------------------|-----------------|
| [Teams.Pokemon]     | [Pokemon.ID]    |
| [Teams.Trainer]     | [Trainers.Name] |
| [Pokemon.Locations] | [Location.Name] |

Tab. 3: The result table for the DPQL query from List. 1 with column combinations X and Y.

Finding the most effective strategy for obtaining DPQL query results will be subject to extensive future research, but a possible way of processing the foreign-key query with state-of-the-art algorithms is as follows: We first discover all INDs, which are  $\{Pokemon\} \subseteq \{ID\}$ ,  $\{Trainer\} \subseteq \{Firstname\}$ ,  $\{Location\} \subseteq \{Title\}$ , and  $\{Strong\} \subseteq \{Biome\}$ ; then we discover all UCCs, which are  $\{ID\}$ ,  $\{Weight\}$ ,  $\{Name, Sex\}$ ,  $\{Name, Size\}$ ,  $\{Title\}$ ,  $\{Firstname\}$ ,  $\{Rank\}$ ,  $\{Pokecount\}$ ; after obtaining both type-specific profiling results, we intersect the IND right-hand-sides and the UCCs with a subset-aware comparison (i. e., if any subset of a right-hand-side is a UCC, the IND-UCC-pair is valid); this leaves us with the INDs shown in Tab. 3; finally, we apply the size and origin filters, which do not change the results. This process demonstrates that DPQL queries can be answered automatically with state-of-the-art profiling technology, although this way of processing is terribly expensive.

### Normalization

While the foreign-key example is an ideal case of a result table, the relational result structure for data profiling statements has a major size issue when it comes to more complex result sets: Because every row in the table represents a unique valid result, DPQL queries with more than one dependency in the output column combinations generate a lot of redundancy. For illustration purposes, consider the DPQL query in List. 5 that aims to profile all unique column combinations in the relations Pokemon and Trainers. Because these UCCs are associated with two independent CC() calls, every combination of a Pokemon UCC and a Trainers UCC is a valid answer to the query. We show the list of results in Tab. 4.

```

1  SELECT
2    X AS PokemonUCCs,
3    Y AS TrainersUCCs
4  FROM
5    CC(Pokemon) X,
6    CC(Trainers) Y
7  WHERE
8    UCC(X)
9    AND UCC(Y)
    
```

List. 5: Find all UCCs in Pokemon and Trainers.

| PokemonUCCs | TrainersUCCs |
|-------------|--------------|
| {ID}        | {Firstname}  |
| {ID}        | {Rank}       |
| {ID}        | {Pokecount}  |
| {Name, Sex} | {Firstname}  |
| {Name, Sex} | {Rank}       |
| {Name, Sex} | {Pokecount}  |
| ...         | ...          |

Tab. 4: UCCs of List. 5 in one result.

The redundancy that we find in this result table can be described as a *join* or *multivalued dependency* [Ab18]. The redundancy introduced with such dependencies grows quadratically with increasing data volume, which is problematic considering that data profiling result sets grow exponentially with the input schema sizes – even the increased pruning capabilities of DPQL cannot resolve this general issue.

```

1  SELECT
2    X, Y, Z
3  FROM
4    CC(Pokemon,Trainers,Teams) X,
5    CC(Pokemon,Trainers,Teams) Y,
6    CC(Pokemon,Trainers,Teams) Z
7  WHERE
8    IND(X, Y)
9    AND FD(Y, Z)

```

List. 6: Find all INDs that point to FDs.

In a first solution attempt, we might reject DPQL queries with non-correlated column combinations, but the redundancy issue also exists for properly correlated column combinations: The DPQL query in List. 6 asks for all inclusion dependencies that point to functional dependencies; the result should list both the INDs and FDs. Now, if an IND points to multiple FDs or an FD is the target of multiple INDs, we generate duplicate, i. e., redundant IND and FD outputs, respectively. So, we again observe redundancy from *join* or *multivalued dependencies* in the results. To resolve these dependency-caused redundancies, relational database theory suggests schema normalization. For this reason, we propose normalized outputs for DPQL queries and, hence, potentially multiple result tables. The algorithm for creating these tables is shown in Algorithm 1. It creates a table for every pair of column combinations that appears together in at least one binary dependency, such as an IND or FD (Lines 3-6); then, it creates separate tables for individual column combinations that are not linked to other column combinations (Lines 7-10). In this way, DPQL results can be represented without their inherent redundancy.

---

#### Algorithm 1 Creation of the normalized DPQL result schema

---

```

1: procedure CREATERESULTSCHEMA(dpqlQuery)
2:   resultSchema ← ∅
3:   for every binary dependency  $D(X, Y)$  in the WHERE clause of the dpqlQuery do
4:     if there is no table  $T(X, Y)$  or  $T(Y, X)$  with  $D$ 's two CCs  $X$  and  $Y$  in resultSchema then
5:       if both  $X$  and  $Y$  are selected in the SELECT clause then
6:         create the table  $T(X, Y)$  and store it in resultSchema
7:   for every CC  $Z$  in the SELECT clause of the dpqlQuery do
8:     if there is no table  $T(Z)$ ,  $T(X, Z)$  or  $T(Z, X)$  with this CC in resultSchema then
9:       if  $Z$  is selected in the SELECT clause then
10:        create the table  $T(Z)$  and store it in resultSchema
11:   return resultSchema

```

---

With normalization, the output of the DPQL query in List. 1 remains one table with schema  $\{\{\text{ForeignKey}, \text{Key}\}\}$ . The output of the DPQL query in List. 5, though, becomes  $\{\{\text{PokemonUCCs}\}, \{\text{TrainerUCCs}\}\}$  and the output of the DPQL query in List. 6 becomes  $\{\{X, Y\}, \{Y, Z\}\}$ . To reconstruct the single, not-normalized result table or to read a full result

row, we simply join the individual result tables on common column combinations (e. g.  $Y$  for the query in List. 6); the reconstruction of unrelated result tables requires a cross join.

In summary, we recommend normalizing DPQL query results for result compaction. The non-normalized, single relation results can always be obtained by joining the result tables, which is useful, for instance, if a DPQL query is embedded into an SQL query.

## Extension Columns

Many data profiling results, such as functional and inclusion dependencies, simply mark a special relation between column combinations. These relations can be expressed with the (normalized) relational result format on column combinations. However, the data profiling toolbox offers a plethora of metadata statements, relaxations, and conditions that provide additional information about the properties of a column combination or column combination relationship. Therefore, we propose to extend DPQL result schemata dynamically with additional columns that store well-defined, metadata-dependent information. The rationale here is simple: If a DPQL function, such as  $\text{UCC}()$ ,  $\text{FD}()$ , or  $\text{IND}()$ , extracts more than a relationship of column combinations, the DPQL engine adds a DPQL function-specific, additional column to the output schema. In theory, we can assume that the union of all possible extension columns is implicitly present in all DPQL query results and the fields are NULL by default, but in practice, these columns should be hidden if they are empty. We now briefly introduce some basic extension columns for popular data profiling metrics (see Tab. 5). It is worth noting that the table is incomplete and needs to be extended in the development process of the data profiling engine:

| Name        | Property   | Type           | Values   |
|-------------|------------|----------------|--|
| Approximate | Relaxation | <i>boolean</i> | true if validity is not certain                        |
| Partial     | Relaxation | <i>float</i>   | <i>Fraction</i> of records that fulfill the statement  |
| Conditional | Relaxation | <i>string</i>  | <i>Condition</i> for defining the statement's scope    |
| Minimum     | Statistic  | <target type>  | <i>Minimum value</i> of the target CC                  |
| Maximum     | Statistic  | <target type>  | <i>Maximum value</i> of the target CC                  |
| Histogram   | Statistic  | <i>string</i>  | <i>Value distribution</i> in the target CC             |
| Denial      | Special    | <i>string</i>  | <i>Denial constraint expression</i> on the target CC   |
| Matching    | Special    | <i>string</i>  | <i>Matching dependency expression</i> on the target CC |
| Order       | Special    | <i>string</i>  | <i>Order dependency expression</i> on the target CC    |

Tab. 5: Extension columns that relax metadata statements or belong to special metadata statements.

**Relaxation:** Any relational metadata statement can be relaxed in different ways [CDP16]: We can, i. a., make the statement *approximate* signaling that the statement's validity is not guaranteed, *partial* to restrict the statement's validity to a certain percentage of records, or *conditional* to tie the statement's validity to specific constraints. Such relaxations have been implemented for many data profiling algorithms and are required by many data profiling applications. With the extension columns, we can also return them in DPQL results.

**Statistic:** Data profiling often targets basic statistics, such as a column combination’s min-, max-, avg-, or median-values, NULL-counts, data types, histograms, lengths- and size-measurements, or frequent item sets. The results of such profiling tasks can easily be stored in extension columns.

**Special:** Some special data dependencies, which are frequently extensions of functional dependencies, can describe more complex relationships between column combinations. This includes, for example, *matching dependencies* (MDs) [Fa08], *order dependencies* (ODs) [GH83], and *denial constraints* (DCs) [Be11]. The extra information hidden in these relationships can be similarity functions and thresholds (see MDs), order directions (see ODs), or entire first-order logic statements (see DCs). While the column combinations of these dependencies are stored in the normal CC-columns of the relational DPQL result sets, the engine adds extension columns for the dependency-specific details.

As shown in Tab. 5, extension columns can be typed to improve their accessibility for applications. The standard for basically all existing data profiling tools and algorithms is to provide all results as strings; therefore, typed extension columns can add some additional information. The concept of extension columns adapts well to the dynamic nature of data profiling, as it allows a flexible combination of properties. For example, Tab. 6 shows the result of a DPQL query that discovered all *partial, conditional matching dependencies*. Although no existing data profiling algorithm can actually discover such dependencies, there is certainly a practical use for them in, for instance, data integration. The result table lists the two column combinations of this dependency (Pokemon and PoMos), the matching dependency condition (Matching), and the two relaxations (Partial and Conditional) in one relational table. Each entry in this result relation – in this case, only one entry – is an answer to the discovery query. The shown example describes the matching dependency  $\{Pokemon[Name] \approx_{Jac,0.92} PoMos[ID]\} \rightarrow \{Pokemon[Sex] \approx_{Lev,1.0} PoMos[Gender]\}$ , which is true for 97% of the tuples under the condition  $\{Weight > 0 \wedge Name \neq 'Mewtwo'\}$ .

| Pokemon                        | PoMos                       | Partial | Conditional                                     | Matching                                    |
|--------------------------------|-----------------------------|---------|---|---|
| [Pokemon.Name,<br>Pokemon.Sex] | [PoMos.ID,<br>PoMos.Gender] | 0.97    | $\{Weight > 0 \wedge$<br>$Name \neq 'Mewtwo'\}$ | $[(Jaccard, 0.92),$<br>$(Levenstein, 1.0)]$ |

Tab. 6: Result of a DPQL query that discovered all *partial, conditional matching dependencies*.

## 6 DPQL Functions

The purpose of DPQL is to restrict data profiling activities and their results in such a way that only truly needed metadata is delivered to the application. To filter and combine the column combinations purposively, DPQL offers a variety of functions that are applied in the **WHERE** clause. This section introduces the most important DPQL functions.

## Metadata Discovery Functions

Metadata discovery functions are the core of DPQL. These functions represent the data profiling services that were traditionally implemented as separate algorithms. In this paper, we already used three metadata discovery functions in the various examples, namely **UCC**(*<CC>*), **FD**(*<CC>*, *<CC>*), and **IND**(*<CC>*, *<CC>*) for unique column combinations, functional dependencies, and inclusion dependencies, respectively. The reading order of column combinations in dependency functions is from left to right: First, the dependent/cause/included part, then the referenced/effect/containing part. Throughout this paper, we showed example queries with UCCs, FDs, and INDs, but thanks to extension columns (see Sect. 5.3), the functional concept extends seamlessly to all other types of metadata, such as order dependencies (**OD**(*<CC>*, *<CC>*)), matching dependencies (**MD**(*<CC>*, *<CC>*)), or denial constraints (**DC**(*<CC>*, *<CC>*)). Short DPQL queries with a single metadata function call can be used as an interface for existing data profiling algorithms, but the strength of DPQL lies in the combination of metadata functions. With a good understanding of the discovery functions, a query engine can combine multiple functions into a single, holistic profiling task and, then, optimize execution orders, share intermediate results for additional search space pruning, and re-use temporary data structures.

To support possibly all variations of data profiling, we need a standard to pass optional configuration parameters to metadata functions. For example, suppose we want to relax a dependency as discussed in Sect. 5.3 or force the declarative query into a certain execution strategy, which is to bypass the automatic query optimizer. In such cases, we can specify these objectives as parameters in the metadata functions. Passing parameters to DPQL functions is done via named parameters with the *<parameter>=<value>* syntax. This syntax ensures that parameter specifications are order-invariant and differ from column combination specifications. To enforce, for instance, a partial functional dependency that has at least a coverage of 95% of the tuples, we could write **FD**(*X*, *Y*, *partial=0.95*) or to force the engine to discover approximate INDs with the FAIDA method [Kr17b], we write **IND**(*X*, *Y*, *approximate=true*, *method='FAIDA'*). We acknowledge that this is not the most idiomatic approach for a declarative query, but it addresses the variety of data profiling demands and the fact that data profiling is still a quickly evolving discipline.

## Result Restriction Functions

**CARDINALITY:** The **CARDINALITY**(*<CC>*) function counts the number of distinct values in a column combination. It can be used together with numeric comparators (i. e., *<*, *<=*, *=*, *>=*, *>*) in filter statements to restrict valid column combinations to those that have a certain (minimum or maximum) cardinality. We have seen this function already in the query of Tab. 1, where we demanded foreign-keys to hold at least two different values.

**SIZE:** The **SIZE(<CC>)** function can be used to restrict the number of attributes in a column combination to a fixed, minimum or maximum size. Recall that the **CC()** function creates a (virtual) power-set-shaped lattice of column combinations of various sizes. With the **SIZE()** function and a numeric comparator (i. e.,  $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$ ), we can bind the sizes of certain column combinations to numeric values or the sizes of other column combinations (see List. 7). In this way, **SIZE()** effectively prunes the search space with a simple criterion that many profiling algorithms can already process.

```

1  SELECT
2    X, Y
3  FROM
4    CC(Pokemon) X,
5    CC(Pokemon) Y,
6    CC(Teams) Z
7  WHERE
8    FD(X,Y) AND SIZE(X) < 3
9    AND IND(Y,Z) AND SIZE(Y) = SIZE(X)

```

List. 7: Find FDs with less than three attributes in Pokemon that functionally determine an IND of same size into the Teams relation.

**MIN** and **MAX:** To keep metadata results concise, data profiling algorithms discover only result sets of minimal (UCCs, FDs, MDs, ...) or maximal (INDs) dependencies; via dependency axioms, all non-enumerated dependencies can be derived from these sets. Now that we combine dependencies into patterns via DPQL, minimality/maximality properties are less clear. Consider, for example, the query in List. 8. The answer to this query might be an  $(X, Y)$ -tuple, where neither  $Y$  is a minimal UCC

```

1  SELECT
2    X AS ForeignKey, Y AS Key
3  FROM
4    CC(Pokemon,Teams,Trainers) X,
5    CC(Pokemon,Teams,Trainers) Y
6  WHERE
7    UCC(Y) AND IND(X,Y) AND MIN(X)

```

List. 8: Find foreign-key candidates with attribute sets of minimal size – effectively unary INDs.

nor  $X \subseteq Y$  is a maximal IND – this is what makes the traditional application of data profiling results such a hard task. To lead the results in a useful and clear direction, we can define specific CCs to be **MIN(<CC>)** or **MAX(<CC>)**. Minimizing means that we cannot remove a single attribute from the CC without violating the entire query result; maximizing means that we cannot add any further attribute. By default, all profiling functions but **IND()** produce minimal results; **IND()** and combinations with this function produce maximal results.

**CONTAINS:** The DPQL function **CONTAINS(<CC>, <CC>)** specifies that in every valid result, the first column combination contains all attributes of the second column combination. To understand the usefulness of this function, again consider the example query in List. 8. Assume we want, as an output of this query, not the actual IND-UCC-pair that answers the query but instead the maximal IND and minimal UCC that frame these solutions. Hence, we specify three outputs  $X$ ,  $Y$ , and  $Z$  constrained to **UCC(X)**, **IND(Y, Z)**, and **CONTAINS(Y, X)**, then  $X \subseteq Y$  (referring to attributes here; not INDs!) with minimal  $X$  and maximal  $Y$  and  $Z$  column combinations.

**SPLIT** and **PAIR**: Column combinations in DPQL query results are, by default, unrelated unless some metadata function connects them. Occasionally, however, we want to filter results such that certain column combinations in a result (= a row in the result table) are either *paired* (= potentially different attribute lists but from the same relation) or *split* (= different attribute lists from different relations). The **PAIR**(*<CC>*, *<CC>*) and **SPLIT**(*<CC>*, *<CC>*)

functions allow the user to specify these requirements in a DPQL query. We recall that in our introductory example on foreign-key discovery (see List. 1) the source and target columns should stem from different relations; this was ensured with the **SPLIT**() function. If we would like to discover, for example, redundant keys (= more than one UCC in the same table) in multiple relations (see List. 9), we need the **PAIR**() function to co-locate X and Y. Note that **PAIR**() and **SPLIT**() are commutative operations, so that e. g. **PAIR**(X, Y) = **PAIR**(Y, X).

```

1  SELECT
2    X, Y
3  FROM
4    CC(Pokemon, Teams, Trainers) X,
5    CC(Pokemon, Teams, Trainers) Y
6  WHERE
7    UCC(X) AND UCC(Y) AND PAIR(X, Y)

```

List. 9: Find redundant keys in multiple relations.

## 7 DPQL in Practical Applications

Data profiling has many applications in data management and data analytics. To evaluate our data profiling query language, we selected a few representative scenarios from different applications to showcase the implementation of their profiling activities in our novel dialect.

**Data linkage:** Our foreign-key discovery example from List. 1 has been drawn from a data engineering task that aims to connect previously unconnected datasets or datasets for which the foreign-key relationships have been lost. The discovered combinations of INDs and UCCs present an application with structurally valid constraint candidates.

**Data cleaning:** Metadata is an important asset for error detection and correction. A cleaning system could, for instance, issue the DPQL query in List. 10 to discover partial INDs, UCCs, and FDs in the *Pokemon* relation. The system would then check the results for meaningful but not 100% correct results. We should, for example, find the IND  $Location \subseteq Title$ , the UCC  $\{Name, Sex\}$ , and the FD  $Type \rightarrow Weak$ . If one of these is indeed partial and not exact, we can use the dependency to identify and possibly correct the erroneous records [MA20].

```

1  SELECT
2    W, X, Y, Z
3  FROM
4    CC(Pokemon) V, CC(Pokemon) W,
5    CC(Pokemon) X, CC(Pokemon) Y,
6    CC(Locations) Z
7  WHERE
8    IND(V, Z, partial=0.8)
9    AND UCC(W, partial=0.95)
10   AND FD(X, Y, partial=0.90)

```

List. 10: Find partial dependencies in the *Pokemon* relation for error detection.

**Query optimization:** Research on optimizing SQL queries with profileable metadata has generated many approaches, ranging from various query rewriting strategies over physical execution optimization techniques to cost-based query plan rewriting rules [Ko22]. For illustration purposes, consider an example of a distinct semi-join filter, which is an SQL query of the form **SELECT DISTINCT** Name, Sex **FROM** Pokemon **WHERE** Location **IN** {**SELECT** Title **FROM** Locations}; The DPQL query in List. 11 checks if {Name, Sex} is unique to remove the **DISTINCT** operator and if  $Location \subseteq Title$  is an IND to remove the entire **WHERE** clause.

```

1 SELECT
2   UCC_C, IND_L, IND_R
3 FROM
4   CC(Pokemon.Name, Pokemon.Sex) UCC_C,
5   {[Pokemon.Location]} AS IND_L,
6   {[Locations.Title]} AS IND_R
7 WHERE
8   UCC(UCC_C) AND IND(IND_L, IND_R)

```

List. 11: Find a specific UCC and IND.

**Data integration:** Schema matching is an integral part of data integration. One flavor of schema matching are structure-based approaches [RB01]. The partial conditional matching dependency that we discussed in Sect. 5.3 is such a structure that describes matching attributes (the MD) with some failure tolerance and context information (the conditional properties). To discover the partial conditional MDs between the Pokemon and PoMos relations, we can use the DPQL query in List. 12.

```

1 SELECT
2   X, Y
3 FROM
4   CC(Pokemon, PoMos) X,
5   CC(Pokemon, PoMos) Y
6 WHERE
7   MD(X, Y, partial=0.95,
8     conditional=true)
9   AND SPLIT(X, Y)

```

List. 12: Find partial conditional MDs.

**Data exploration:** A look at the metadata of a relational dataset often helps to understand its structure and implicit logic better. Because data often comes without metadata, data profiling is conducted to gather possibly many insights from a given instance. The DPQL query in List. 13 does exactly this: It collects all UCCs, FDs, and INDs that are true in our Pokémon example. The normalized output presents the results in three tables – one for each dependency.

```

1 SELECT
2   UCC_C, FD_L, FD_R, IND_L, IND_R
3 FROM
4   CC(P,L,T,T) UCC_C,
5   CC(P,L,T,T) FD_L, CC(P,L,T,T) FD_R,
6   CC(P,L,T,T) IND_L, CC(P,L,T,T) IND_R
7 WHERE
8   UCC(UCC_C) AND FD(FD_L, FD_R)
9   AND IND(IND_L, IND_R)

```

List. 13: Find UCCs, FDs and INDs in all tables; table names in CC() calls were shortened for brevity.

## 8 Future Work

The data profiling query language (DPQL) that we introduced in this paper is an essential building block for a new generation of data profiling systems. For its practical implementation, we envision a database-like system that covers all standard query processing components. The setup of this system, however, is more like a *virtually integrated database* [DHI12] or, in modern terms, a *DataLakehouse* [Ar21], because the system answers metadata questions in a virtual fashion, across potentially multiple datasets, and without manipulating the data itself. Due to the size and complexity of this system, we expect that DPQL will spark innovative research on at least the following components:

**Query parser:** The DPQL queries require a parsing component that translates them into logical (and physical) execution plans. We assume that DPQL can be combined with SQL, but this imposes interesting parsing challenges. Another challenge for the parser (and all other components) is that further iterations of DPQL must be able to introduce new features, such as additional metadata types, metadata properties, or filter functions, because data profiling – in contrast to relational query processing – is a still evolving area.

**Query optimizer:** Despite their similarities with SQL queries, DPQL queries translate into quite different execution plans, for which other optimization rules apply. For query optimization, novel approaches for indexing, caching, query rewriting, operator ordering etc. need to be found. Effective approaches for selecting the most efficient execution strategy (e. g. UCCs first, INDs first, or both at the same time?) and automatically inferring empty results (e. g. from `CC(Pokemon, !Pokemon)` or `SIZE(CC(Team))>3`) are crucial for the system.

**Query execution engine:** The actual data profiling might change significantly given the new application-specific pruning rules and the potential of holistically combining profiling runs. Given the many existing profiling algorithms (and new techniques of the future), research will need to investigate which algorithms to combine, how to combine algorithms, and how to integrate them into one system. Considering the comprehensive amount of metadata types and discovery flavors, we expect a lot of future research on the actual query processing.

## 9 Summary

In this paper, we proposed DPQL, a declarative query language for the discovery of data dependencies and other metadata statements. DPQL is the first uniform data profiling language and an essential building block for a new generation of data profiling systems. The SQL-like language relieves data scientists from deploying complex profiling algorithms, and it renders most of the expensive and difficult post-processing efforts obsolete. Due to the increased filter- and pruning-capabilities, we expect significant efficiency gains for DPQL-based data profiling activities. With DPQL, we started to close the gap between data profiling results and actual applications needs; now, much research is needed for the technical design of the language and its profiling capabilities.

## References

- [Ab18] Abedjan, Z.; Golab, L.; Naumann, F.; Papenbrock, T.: *Data Profiling: Synthesis Lectures on Data Management*. Morgan & Claypool Publishers, 2018.
- [Ar21] Armbrust, M.; Ghodsi, A.; Xin, R.; Zaharia, M.: *Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics*. In: *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 2021.
- [Be11] Bertossi, L. E.: *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.
- [Bi20] Birnick, J.; Bläsius, T.; Friedrich, T.; Naumann, F.; Papenbrock, T.; Schirneck, M.: *Hitting Set Enumeration with Partial Information for Unique Column Combination Discovery*. *Proceedings of the VLDB Endowment* 13/12, pp. 2270–2283, 2020.
- [BKN17] Bleifuß, T.; Kruse, S.; Naumann, F.: *Efficient Denial Constraint Discovery with Hydra*. *Proceedings of the VLDB Endowment* 11/3, pp. 311–323, 2017.
- [Bl12] Blockeel, H.; Calders, T.; Fromont, É.; Goethals, B.; Prado, A.; Robardet, C.: *An inductive database system based on virtual mining views*. *Data Mining and Knowledge Discovery* 24/1, pp. 247–287, 2012.
- [Ca21a] Caruccio, L.; Cirillo, S.; Deufemia, V.; Polese, G.: *Efficient Discovery of Functional Dependencies from Incremental Databases*. In: *International Conference on Information Integration and Web Intelligence*. Pp. 400–409, 2021.
- [Ca21b] Caruccio, L.; Deufemia, V.; Naumann, F.; Polese, G.: *Discovering Relaxed Functional Dependencies Based on Multi-Attribute Dominance*. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 33/9, pp. 3212–3228, 2021.
- [CDP16] Caruccio, L.; Deufemia, V.; Polese, G.: *Relaxed Functional Dependencies - A Survey of Approaches*. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 28/1, pp. 147–165, 2016.
- [Ch17] Chardin, B.; Coquery, E.; Pailloux, M.; Petit, J.-M.: *RQL: a query language for rule discovery in databases*. *Theoretical Computer Science* 658/, pp. 357–374, 2017.
- [Ch19] Chen, H.; Jajodia, S.; Liu, J.; Park, N.; Sokolov, V.; Subrahmanian, V. S.: *FakeTables: Using GANs to Generate Functional Dependency Preserving Tables with Bounded Real Data*. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. International Joint Conferences on Artificial Intelligence Organization, pp. 2074–2080, 2019.
- [De22] *Desbordante: Open-source data profiling tool*, 2022, URL: <https://desbordante.unidata-platform.ru/>, visited on: 09/19/2022.

- [DHI12] Doan, A.; Halevy, A.; Ives, Z.: Principles of data integration. Elsevier, 2012.
- [DR02] Do, H.-H.; Rahm, E.: COMA – A System for flexible combination of schema matching approaches. In: Proceedings of the International Conference on Very Large Databases (VLDB). Pp. 610–621, 2002.
- [Dü19] Dürsch, F.; Stebner, A.; Windheuser, F.; Fischer, M.; Friedrich, T.; Strelow, N.; Bleifuß, T.; Harmouch, H.; Jiang, L.; Papenbrock, T.; Naumann, F.: Inclusion Dependency Discovery: An Experimental Evaluation of Thirteen Algorithms. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM). Pp. 219–228, 2019.
- [Eh16] Ehrlich, J.; Roick, M.; Schulze, L.; Zwiener, J.; Papenbrock, T.; Naumann, F.: Holistic Data Profiling: Simultaneous Discovery of Various Metadata. In: Proceedings of the International Conference on Extending Database Technology (EDBT). Pp. 305–316, 2016.
- [Fa08] Fan, W.: Dependencies Revisited for Improving Data Quality. In: Proceedings of the Symposium on Principles of Database Systems (PODS). Pp. 159–170, 2008.
- [Fe18] Fernandez, R. C.; Abedjan, Z.; Koko, F.; and Samuel Madden, G. Y.; Stonebraker, M.: AURUM: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE). 2018.
- [FL10] Fang, L.; LeFevre, K.: Splash: ad-hoc querying of data and statistical models. In: Proceedings of the International Conference on Extending Database Technology (EDBT). Pp. 275–286, 2010.
- [GH83] Ginsburg, S.; Hull, R.: Order dependency in the relational model. *Theoretical Computer Science* 26/1–2, pp. 149–195, 1983.
- [HL18] Hannula, M.; Link, S.: On the interaction of functional and inclusion dependencies with independence atoms. In: International Conference on Database Systems for Advanced Applications. Pp. 353–369, 2018.
- [HN17] Harmouch, H.; Naumann, F.: Cardinality Estimation: An Experimental Survey. *Proceedings of the VLDB Endowment* 11/4, pp. 499–512, 2017.
- [HPN21] Harmouch, H.; Papenbrock, T.; Naumann, F.: Relational Header Discovery using Similarity Search in a Table Corpus. In: Proceedings of the International Conference on Data Engineering (ICDE). Pp. 444–455, 2021.
- [Hu99] Huhtala, Y.; Kärkkäinen, J.; Porkka, P.; Toivonen, H.: TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal* 42/2, pp. 100–111, 1999.
- [IC15] Ilyas, I. F.; Chu, X.: Trends in Cleaning Relational Data: Consistency and Deduplication. *Foundations and Trends in Databases* 5/4, pp. 281–393, 2015.
- [Ko22] Kossmann, J.; Lindner, D.; Naumann, F.; Papenbrock, T.: Workload-driven, lazy discovery of data dependencies for query optimization. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR). 2022.

- [KPN20] Koumarelas, I.; Papenbrock, T.; Naumann, F.: MDedup: Duplicate Detection with Matching Dependencies. 13/5, pp. 712–725, 2020.
- [KPN22] Kossmann, J.; Papenbrock, T.; Naumann, F.: Data dependencies for query optimization: a survey. *The VLDB Journal* 31/1, pp. 1–22, 2022.
- [Kr16] Kruse, S.; Papenbrock, T.; Harmouch, H.; Naumann, F.: Data Anamnesis: Admitting Raw Data into an Organization. *IEEE Data Engineering Bulletin* 39/2, pp. 8–20, 2016.
- [Kr17a] Kruse, S.; Hahn, D.; Walter, M.; Naumann, F.: Metacrate: Organize and analyze millions of data profiles. In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. Pp. 2483–2486, 2017.
- [Kr17b] Kruse, S.; Papenbrock, T.; Dullweber, C.; Finke, M.; Hegner, M.; Zabel, M.; Zoellner, C.; Naumann, F.: Fast Approximate Discovery of Inclusion Dependencies. In: *Proceedings of the Conference Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*. Pp. 207–226, 2017.
- [Li20] Livshits, E.; Heidari, A.; Ilyas, I. F.; Kimelfeld, B.: Approximate Denial Constraints. *Proceedings of the VLDB Endowment* 13/10, pp. 1682–1695, 2020.
- [LSS96] Lakshmanan, L. V. S.; Sadri, F.; Subramanian, I. N.: SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. In: *Proceedings of the VLDB Endowment*. Pp. 239–250, 1996.
- [MA20] Mahdavi, M.; Abedjan, Z.: Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proceedings of the VLDB Endowment* 13/12, pp. 1948–1961, 2020.
- [MPC+96] Meo, R.; Psaila, G.; Ceri, S., et al.: A new SQL-like operator for mining association rules. In: *Proceedings of the International Conference on Very Large Databases (VLDB)*. Vol. 96, pp. 122–133, 1996.
- [MPC98] Meo, R.; Psaila, G.; Ceri, S.: An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery* 2/2, pp. 195–224, 1998.
- [OP11] Ordonez, C.; Pitchaimalai, S. K.: One-pass data mining algorithms in a DBMS with UDFs. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*. Pp. 1217–1220, 2011.
- [Pa00] Paulley, G. N.: *Exploiting Functional Dependence in Query Optimization*, tech. rep., University of Waterloo, 2000.
- [Pa15a] Papenbrock, T.; Bergmann, T.; Finke, M.; Zwiener, J.; Naumann, F.: Data Profiling with Metanome. *Proceedings of the VLDB Endowment* 8/12, pp. 1860–1863, 2015.
- [Pa15b] Papenbrock, T.; Ehrlich, J.; Marten, J.; Neubert, T.; Rudolph, J.-P.; Schönberg, M.; Zwiener, J.; Naumann, F.: Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proceedings of the VLDB Endowment* 8/10, pp. 1082–1093, 2015.

- [Pa15c] Papenbrock, T.; Kruse, S.; Quiané-Ruiz, J.-A.; Naumann, F.: Divide & Conquer-based Inclusion Dependency Discovery. Proceedings of the VLDB Endowment 8/7, pp. 774–785, 2015.
- [PAN21] Pena, E. H. M.; de Almeida, E. C.; Naumann, F.: Fast Detection of Denial Constraint Violations. Proceedings of the VLDB Endowment 15/4, pp. 859–871, 2021.
- [PN16] Papenbrock, T.; Naumann, F.: A Hybrid Approach to Functional Dependency Discovery. In: Proceedings of the International Conference on Management of Data (SIGMOD). Pp. 821–833, 2016.
- [RB01] Rahm, E.; Bernstein, P. A.: A survey of approaches to automatic schema matching. Proceedings of the VLDB Endowment 10/4, pp. 334–350, 2001.
- [Ro09] Rostin, A.; Albrecht, O.; Bauckmann, J.; Naumann, F.; Leser, U.: A Machine Learning Approach to Foreign Key Discovery. In: Proceedings of the ACM Workshop on the Web and Databases (WebDB). 2009.
- [Sc20] Schirmer, P.; Papenbrock, T.; Koumarelas, I.; Naumann, F.: Efficient Discovery of Matching Dependencies. ACM Transactions on Database Systems (TODS) 45/3, pp. 1–33, 2020.
- [SGI19] Saxena, H.; Golab, L.; Ilyas, I. F.: Distributed Implementations of Dependency Discovery Algorithms. Proceedings of the VLDB Endowment 12/11, pp. 1624–1636, 2019.
- [SP22] Schmidl, S.; Papenbrock, T.: Efficient Distributed Discovery of Bidirectional Order Dependencies. VLDB Journal 31/1, pp. 49–74, 2022.
- [Sz17] Szlichta, J.; Godfrey, P.; Golab, L.; Kargar, M.; Srivastava, D.: Effective and Complete Discovery of Order Dependencies via Set-based Axiomatization. Proceedings of the VLDB Endowment/, 2017.
- [VA18] Visengeriyeva, L.; Abedjan, Z.: Metadata-Driven Error Detection. In: Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM). 2018.
- [Vi22] Viadotto: Make your data profitable with our next-gen data profiling tools, 2022, URL: <https://www.viadotto.tech/>, visited on: 09/19/2022.
- [Wa17] Wang, Y.; Song, S.; Chen, L.; Yu, J. X.; Cheng, H.: Discovering conditional matching rules. ACM Transactions on Knowledge Discovery from Data 11/4, pp. 1–38, 2017.
- [WHL21] Wei, Z.; Hartmann, S.; Link, S.: Algorithms for the discovery of embedded functional dependencies. VLDB Journal 30/6, pp. 1069–1093, 2021.
- [WLL19] Wei, Z.; Leck, U.; Link, S.: Discovery and Ranking of Embedded Uniqueness Constraints. Proceedings of the VLDB Endowment 12/13, pp. 2339–2352, 2019.

- [Xi22] Xiao, R.; Yuan, Y.; Tan, Z.; Ma, S.; Wang, W.: Dynamic Functional Dependency Discovery with Dynamic Hitting Set Enumeration. In: Proceedings of the International Conference on Data Engineering (ICDE). Vol. 1. 1, pp. 286–298, 2022.
- [Zh10] Zhang, M.; Hadjieleftheriou, M.; Ooi, B. C.; Procopiuc, C. M.; Srivastava, D.: On Multi-column Foreign Key Discovery. Proceedings of the VLDB Endowment 3/1-2, pp. 805–814, 2010.



# ExtracTable: Extracting Tables from Raw Data Files

Leonardo Hübscher,<sup>1</sup> Lan Jiang,<sup>2</sup> Felix Naumann<sup>3</sup>

**Abstract:** Raw data, especially in text-files, comes in many shapes and forms, often tailored toward human readability. They include preambles and footnotes, are formatted visually, and in general do not follow csv-guidelines. The ability to easily ingest such files into data systems opens up many opportunities for data analysis and processing. With `EXTRACTABLE`, we present a system that can automatically ingest a large variety of raw data files, including text files and poorly structured csv-files by detecting row patterns and thus separating their values into coherent columns. We manually annotated 957 files of a wide variety containing 1 208 tables. We show experimentally that `ExtracTable` can correctly parse 90% of all lines in structured files and 76% of all lines in files with a visual layout only, significantly outperforming state-of-the-art.

## 1 Table Extraction

As more and more data is created and made accessible, the ability to automatically ingest and analyze them becomes increasingly desirable. Various open data portals are a means for governments, companies, and individuals to make data publicly available. However, to support data creators to easily share their data, these platforms do not enforce specific data formats, and we observe very many home-grown formats that are not amenable to easy parsing and ingesting into a data system.

Data wrangling summarizes the process of transforming raw data into a well-defined format. According to multiple studies from Kaggle, Anaconda, IBM, and Forbes, data scientists spend 26% to 80% of their time on data wrangling, distracting them from tackling the original data processing task [An20; Ch14; Mo18; Pr16]. This effort is not only time-consuming, but also tedious and error-prone. Still, data preparation is necessary, as data quality issues otherwise prevent subsequent algorithms from working well.

Data is often displayed and stored in a tabular format that is suitable for both humans and machines. However, tables may appear quite different when persisted as files. Plain-text files, for instance, lack proper instructions on how to interpret tables therein. Our work regards two table formats: csv tables and ASCII tables. The widely used csv (character-separated-values) format was first used by IBM to store tabular data in 1972. However, a formally specified csv format, which is now known as the RFC 4180 standard, had not been formalized until 33 years later [IB72]. Meanwhile, companies and data practitioners have developed their

---

<sup>1</sup> Hasso Plattner Institute, University of Potsdam, Germany leonardo.huebscher@student.hpi.de

<sup>2</sup> Hasso Plattner Institute, University of Potsdam, Germany lan.jiang@hpi.de

<sup>3</sup> Hasso Plattner Institute, University of Potsdam, Germany felix.naumann@hpi.de

```

# OBIA4RTM config file for setting up Prospect4SAIL
#
# Typical values (taken from J Gomez-Dans on https://pypi.org/project/prosail/)
#
# =====
# | Parameter | Description of parameter | Units | Typical min | Typical max |
# |-----|-----|-----|-----|-----|
# | N | Leaf structure parameter | N/A | 0.8 | 2.5 |
# | cab | Chlorophyll a+b concentration | ug/cm2 | 0 | 80 |
# | caw | Equivalent water thickness | cm | 0 | 200 |
# | car | Carotenoid concentration | ug/cm2 | 0 | 20 |
# | cbrown | Brown pigment | NA | 0 | 1 |
# | cm | Dry matter content | g/cm2 | 0 | 200 |
# | lai | Leaf Area Index | N/A | 0 | 10 |
# | lidfa | Leaf angle distribution | N/A | - | - |
# | lidfb | Leaf angle distribution | N/A | - | - |
# | psoil | Dry/Wet soil factor | N/A | 0 | 1 |
# | rsoil | Soil brightness factor | N/A | - | - |
# | hspot | Hotspot parameter | N/A | - | - |
# | tts | Solar zenith angle | deg | 0 | 90 |
# | tto | Observer zenith angle | deg | 0 | 90 |
# | phi | Relative azimuth angle | deg | 0 | 360 |
# | typelidf | Leaf angle distribution type | Integer | - | - |
# =====
#
# You can enter your values below -> make sure not to alter the overall structure of this
# template -> otherwise bad things might happen
#
# Further Explanations:
#
# min: Minimum Value of Parameter
# max: Maximum Value of Parameter (in case min=max, the parameter will not be retrieved)
# num: in case min!=max, the number of samples to be drawn for the specific parameter
# dist: which statistical distribution of values should be used for drawing the samples (igno
# 1: truncated Gaussian (between min and max)
# 2: uniform distribution (between min and max)
# 0: non-applicable
# mean: mean in case of truncated Gaussian distribution
# std: in case of truncated Gaussian standard deviation of parameter for drawing the samples
#
# min max num dist mean std comment
1.8 1.8 1 0 1.5 0 N
20 60 40 1 40 15 cab
0 0 1 0 0 0 car
0 1 10 2 0 0 cbrown
0.01 0.01 1 0 0 0 cw
0.009 0.009 1 0 0 0 cm
0.2 7 40 1 4 2.5 lai
-0.35 -0.35 1 2 0 0 lidfa
-0.15 -0.15 1 0 0 0 lidfb
0.5 0.5 1 0 0 0 rsoil
0.2 0.2 1 0 0 0 psoil
0.01 0.01 1 0 0 0 hspot
27.947 27.947 1 0 0 0 tts
7.04345 7.04345 1 0 0 0 tto
146.691 146.691 1 0 0 0 psi
1 1 1 0 0 0 typelidf

```

Fig. 1: A real-world plain-text file including two tables in different formats (framed in blue) taken from the Mendeley data portal (doi: 10.17632/vs55cwssyh.2#file-54e4f7c2-0156-4be8-9960-d95b0ba0f940)

own formats that use different utility characters, such as “|” as delimiters, which deviate from the specification. Unfortunately, the RFC formalization does not account for such variations. Our previous work recognizes table regions in CSV files with visual features based on different cell data types [VJN21]. To use this approach, however, one must first identify cells. ASCII tables are another type of plain-text data format used to deposit data. Unlike CSV tables that structure data with particular utility characters, ASCII tables merely store characters, leaving the interpretation of file structures to users. The existence of customized file structures forces data scientists to take care of each file individually.

Figure 1 shows the content of a single real-world file with two tables. While the first table uses special characters, such as “|”, “=”, and “-” to frame the header row and different columns, the second table uses whitespace regions to separate columns. To facilitate human readability, columns in the two tables visually align their values by using different numbers of utility characters as field separators. There are also texts before or after tables that typically deliver contextual information, such as experimental setups or sensor information. Texts might be misinterpreted as structured data when they contain table-like elements. Due to the ad-hoc shapes of tables, common commercial tools fail to load them correctly [HN20].

We propose the `EXTRACTABLE` algorithm for automatic table extraction from plain-text files, which takes all the aforementioned file varieties into consideration. Given a file, `EXTRACTABLE` first detects its structure interpretation and uses it to interpret structures of its lines. Then, the algorithm extracts value patterns of the interpreted lines and builds table candidates with the optimal pattern consistency. Finally, a subset of table candidates are selected as the output tables. Our approach makes the following contributions:

1. A set of 957 annotated raw data files selected from a variety of sources, totaling 1 208 tables across all files.
2. The `EXTRACTABLE` approach, which detects column and row patterns in data, and ultimately extracts table elements from raw data files.
3. A detailed experimental evaluation, also comparing to multiple CSV parsing tools and the Pytheas system [Ch20]

To encourage further research on this topic, we have published all annotated data and the code<sup>4</sup>. We organize the rest of this paper as follows: Section 2 summarizes related work. We formalize the terms used in this work and the table extraction problem in Section 3. We elaborate on the proposed `EXTRACTABLE` algorithm in Section 4, and present the results of a series of experiments in Section 5. Finally, we conclude the paper and point out future work in Section 6.

---

<sup>4</sup> <https://github.com/HPI-Information-Systems/ExtracTable>

## 2 Related Work

The Pytheas system addresses the problem of table discovery in csv files using a set of weighted fuzzy rules that exploit column patterns [Ch20]. The weights were trained on a dataset collected from open data portals containing governmental data. The paper focuses on table discovery and row classification. While the authors optimized their approach for csv tables, Pytheas could also be applied to tables in ASCII files if adapted accordingly. A limitation of this approach is that input files must have been parsed properly, which the authors conduct with the standard Sniffer module of Python's csv library in a pre-processing step. In comparison, our approach can parse raw files automatically before detecting tables and classifying rows. We use Pytheas as a baseline for table range detection.

Pyreddy and Croft propose an approach to detect text lines containing tabular structures represented in ASCII [PC97]. A followup work improves the line classification step [Pi03]. From their work, we learn that whitespace alignments in continuous lines are important for ASCII tables. This observation is confirmed by additional related work, such as [SJT03] and [Hu99]. Thus, our approach also makes use of whitespace alignments. However, we note that line classification is only one aspect toward actually extracting tabular data. The original approach relies solely on the structural features of tables and does not take cell content into account, which we consider in our approach.

Döhmen et al. noted that existing csv parsers make decisions during the file parsing process sequentially, which they suspect to negatively impact the overall quality [DMB17]. They propose a solution that makes decisions about sub-criteria as late as possible, trading run time for parsing quality. Besides csv parsing, their heuristics cover file encoding detection, table normalization, and table area detection. While the approach includes a stage dealing with table area detection, it does not handle multi-table files that account for about 7% of the cases in our dataset. Additionally, the authors tested only a limited set of csv variants. We include their published implementation as a baseline when comparing parsing accuracy.

In [BNS19] the authors introduce a novel data consistency measure to correctly parse csv files. The consistency measure consists of a row pattern score and a data type score. The row pattern represents the column count per row, depending on the detected csv dialect. The type score uses regular expressions to detect known data types within cell values and represents the ratio of known cells compared to the total number of cells. Both scores contribute equally to the consistency measure, favoring the pattern score on ties. The pattern-based approach seems to work well according to the provided evaluation. Yet, this solution also does not work with files containing multiple tables that our approach is able to handle. As the authors noted, it can become problematic if many of the cell data types are unknown. We use the publicly available implementation in our experiments.

Ill-formed csv and ASCII files are not the only opportunity to extract relations from content that is designed to be human-readable. For instance, Chu et al. suggest the Tegra approach to recognize relational tables that appear as lists on web pages with the global record alignment

technique [Ch15]. As our approach is not designed to handle web tables, we do not compare to this approach.

Overall, existing works lack at least one of the aforementioned features: 1) parse input files automatically; 2) take content into account; 3) handle multi-table files. Our approach can handle all these limitations.

### 3 Table Formats

We recognize two table formats used for persisting data tables in raw plain-text files: character-separated-values tables (CSV) and other (ASCII) formats. We first introduce these two table formats in detail, and then state the table extraction problem.

#### 3.1 CSV and ASCII tables

According to RFC 4180, a CSV file is a line-wise plain-text file that stores a table: each line represents a data record, and the first line optionally represents the table header [Sh05]. The cells of each line are separated by a special character, the *delimiter*. If a cell value includes the delimiter character itself, the value must be put into quotes using *quotation* characters. An *escape* character is used to escape a quote character or the escape character itself, if they appear within quoted field values. A file's *dialect* specifies the used delimiter  $d$ , quotation  $q$ , and escape characters  $e$ , denoted as  $\langle d, q, e \rangle$  [Sh05]. Although the RFC document specifies comma as delimiter and double quote as quotation and escape, it acknowledges the usage of a wide variety of characters for each dialect component in real-world data [Sh05]. Because CSV files do not carry metadata, the presence of different dialects within and across files acts as a barrier to the automatic table interpretation and extraction.

A W3C working group for “CSV on the Web” proposes the delivery of an additional JSON file, which contains information about the used dialect and the schema [BTH16]. CSVY is a similar development, which stores such information as a YAML meta block at the beginning of the file ([www.csvy.org](http://www.csvy.org)). However, neither standard has been widely adopted.

An **ASCII table** separates columns with white space. To visually align values within each column, ASCII tables fill the column gap between fields with one or more space or tab characters. With their visual alignment, ASCII tables are more suitable for human readability. Fields are separated by white space so that values from different columns do not horizontally interfere each other. Two columns must be separated by at least one whitespace character. Because the characters and their number may vary between different pairs of neighboring fields, we cannot simply delimit lines by using a fixed number of whitespace characters. It is also not possible to accept an arbitrary number of space characters as delimiter, as empty fields would not be recognized properly and field values themselves might include spaces. Instead, a set of *column boundaries* is required: each boundary defines the inclusive start and

exclusive end of a column as the interval  $[start, end)$ , based on the character index. Figure 2 shows an ASCII table using  $[0, 5)$ ,  $[7, 23)$ ,  $[26, 31)$ , and  $[32, 35)$  as column boundaries.

|   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |  |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|--|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |   |   |   |  |
| P | a | r | a | m |   |   | D | e | s | c  | r  | i  | p  | t  | i  | o  | n  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |  |
| - | - | - | - | - | - | - | - | - | - | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | -  | - | - | - |  |
| l | a | i |   |   |   |   | L | e | a | f  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |  |
| l | i | d | f | a |   |   | L | e | a | f  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |  |

Fig. 2: Exemplary ASCII table – columns aligned by layout

ASCII tables can include style information, such as borders, which can make the table structure clearer to human readers. In particular, horizontal lines are often used for underlining headers or separating tables. We therefore distinguish two line types. We refer to a line as a *helper* line if its content includes only non-alphanumeric characters or whitespace. All lines with at least one alphanumeric character are *content* lines.

### 3.2 The Table Extraction Problem

Before we can extract tables from plain-text files, we must understand the structures of these files by identifying the dialects and the column boundaries of tables stored in these two respective file formats. We refer to dialects and column boundaries as *parsing instructions* for the two types of files. Here, we highlight the difficulty resulting from the lack of parsing instructions due to multiple valid ways of interpreting lines.

Figure 3 shows a file excerpt allowing for multiple ASCII interpretations. When regarding the first two lines only, we might split each line into five fields, namely Leaf, angle, distribution, N/A, and -. However, the introduction of the third line yields multiple interpretation possibilities.

|                         |     |   |
|-------------------------|-----|---|
| Leaf angle distribution | N/A | - |
| Leaf angle distribution | N/A | - |
| Dry/Wet soil factor     |     | 0 |

Fig. 3: ASCII table adapted from Figure 1 emphasizing the ambiguity of some ASCII records

Similar effects can be observed for tables stored in the csv format. Even if we consider only dialects using single non-alphanumeric characters, we can generate seven valid delimiter candidates for a line with the text "N, \"Leaf\"structure";N/A;"0.8". A simple approach may select the delimiter character based on the candidate frequency across lines. However, this method is sensitive to the content. For example, cells including delimiter-characters could easily fail this approach.

We state the *table extraction problem from plain-text files* as follows: Given a plain-text file containing one or more vertically stacked tables, determine the line structure and the range (the beginning and the end row indexes) of each table and transform every table to

the RFC 4180 standard. For simplicity, we assume that individual cells do not contain line breaks. Moreover, we exclude the detection of the file encoding from our problem and assume UTF-8 as specified in RFC 4180.

## 4 The EXTRAC<sub>TABLE</sub> algorithm

We propose EXTRAC<sub>TABLE</sub>, an algorithm that exploits data type consistency within columns to tackle the table extraction problem. To interpret fields in a file, EXTRAC<sub>TABLE</sub> first detects per-line valid dialects for CSV tables and possible column boundaries for ASCII tables (Section 4.1). After applying the detected parsing instructions, the resulting interpretations divide each line into several fields, which are passed to the next step to identify data type patterns (Section 4.2). Then our approach generates table candidates with the compatibility score (Section 4.3). Finally, the algorithm selects a subset of the table candidates (Section 4.4).

### 4.1 Parsing instruction detection

Detecting parsing instructions is modeled as dialect detection for CSV tables and column boundary detection for ASCII tables, respectively. EXTRAC<sub>TABLE</sub> first pre-processes a file by classifying each line as either a helper line or a content line. It prunes all helper lines, as they neither deliver content nor help detect correct column boundaries of ASCII tables, and may be incorrectly treated as part of the header or the data region of a table.

**Dialect detection for csv tables.** To recognize a CSV table’s dialect, we propose a two-step approach that first detects all delimiter candidates, and then quotation and escape characters for each delimiter candidate. First, EXTRAC<sub>TABLE</sub> replaces consecutive alphanumeric characters and excluded characters within a line  $l$  with a placeholder character. It then splits the resulting string by the placeholder character, yielding a list of delimiter sequences and empty values. All substring combinations of each delimiter sequence are appended to the list. Values that are empty or longer than the maximum length are removed.

For each detected delimiter, EXTRAC<sub>TABLE</sub> tries to recognize the quotation and escape characters using a depth-first search method, shown in Algorithm 1. The algorithm receives the line content  $l$  and a delimiter sequence  $d$  as input. It tries to parse the line using the dialect  $dialect_0 = \langle d, \varepsilon, \varepsilon \rangle$  (see line 16). The parse method iterates over the character positions of the trimmed line content. For each iteration, `get_dialect_component` returns the component matching the dialect specified in the method parameters following the RFC 4180 grammar. The component can be one of content, delimiter, quotation, escape, or error (see line 5). In cases where the character at the current position *cursor* violates the RFC 4180 grammar, the `get_dialect_component` method returns an error and disregards the dialect (line 7). The state machine for parsing the dialect specified in the *parse* method parameters

is updated in `update_parser_state` (line 12) based on the returned component. Additionally, the algorithm checks whether the remaining line starts a new component from the given dialect. If the current position was classified as content and is not alphanumeric, we could interpret the content as a quotation or escape. Line 9 starts a new branch of the DFS using the remaining line content and the updated dialect  $dialect_1 = \langle d, q, \varepsilon \rangle$ , where  $q$  is the character sequence at the current position. The same logic is applied to the escape character, as shown in line 11. If the parser can process the whole line without errors, it found a legitimate dialect. Finally, the parser returns all valid dialects.

---

**Algorithm 1:** Quotation  $q$  and escape  $e$  character detection
 

---

**Input:** Line content  $l$  and delimiter  $d$

**Output:** A set of dialects

```

1 Def parse( $l, d, q, e, cursor$ ):
2    $cursor=0$ 
3    $tl=trim(l)$ 
4   while  $cursor < |tl|$  do
5      $\langle component, length \rangle = get\_dialect\_component(tl, cursor, d, q, e)$ 
6     if  $component="error"$  then
7       return  $\varepsilon$ 
8     if  $q = \varepsilon \wedge component = "content" \wedge \neg isalnum(tl_{cursor})$  then
9        $parse(tl, d, tl_{cursor}, \varepsilon, cursor)$ 
10    if  $q \neq \varepsilon \wedge e = \varepsilon \wedge component = "content" \wedge \neg isalnum(tl_{cursor})$  then
11       $parse(tl, d, q, tl_{cursor}, cursor)$ 
12       $update\_parser\_state(component)$ 
13       $cursor = cursor + length$ 
14     $dialects = dialects \cup \langle d, q, e \rangle$ 
15  $dialects = []$  // square brackets denote list
16  $parse(l, d, \varepsilon, \varepsilon, 0)$  // start DS using delimiter  $d$ , empty quotation and escape
17
18 return  $dialects \setminus \{\varepsilon\}$ 

```

---

**Column boundary detection for ASCII tables.** Per our definition of ASCII tables, two columns must be separated by a vertical line that has at least one space character. A *vertical line* is a consecutive set of character positions, where all characters in lines are whitespace. To infer the boundaries for all columns, EXTRACTABLE detects vertical lines in-between columns. Algorithm 2 shows the proposed approach to detect vertical lines in an ASCII table. We explain the algorithm using the example depicted in Figure 4. The variable  $k$  depicts the line index. The file *width* is the length of the longest line within a file. The set of whitespace characters is represented by the variable  $WS$ .

**Transform line content into bitmap** with `transform( $l, width$ )`: The algorithm first transforms the line content  $l$  into a bitmap. As lines may contain a combination of tabs and spaces for aligning columns, all tab characters are expanded with the corresponding number of space characters first. We use a tab size of eight, which is the default number in Python’s `expandtabs` function. All whitespace characters are then replaced with 1 (*True*) and any

|         |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
|---------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
|         | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |  |  |
| $k = 0$ | N | a | m | e |   |   |   |   |   | M | i  | n  | u  | t  | e  |    |    |    |    | Q  | u  | o  | t  | e  |    |    |    |    |    |    |    |    |    |    |  |  |
| $k = 1$ | = | = | = | = |   |   |   |   |   | = | =  | =  | =  | =  |    |    |    |    | =  | =  | =  | =  |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| $k = 2$ | V | i | c | t | o | r | i | a |   |   |    |    |    |    |    |    |    |    |    | N  | /  | A  |    |    |    |    |    |    |    |    |    |    |    |    |  |  |
| $k = 3$ | H | a | r | r | y |   |   |   |   |   |    |    |    | 4  | 0  |    |    |    |    | l  | i  | k  | e  |    | c  | o  | m  | p  | l  | e  | x  | .  |    |    |  |  |

(a) An example file with four lines where characters are displayed in monospaced font.

|         |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |
|---------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|
|         | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |   |
| $k = 0$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1 |
| $k = 1$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 0  | 0  | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  |   |
| $k = 2$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 0  | 0  | 0  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  |   |
| $k = 3$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1  | 1  | 1  | 0  | 0  | 1  | 1  | 1  | 1  | 0  | 1  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |   |

(b) A bitmap representation where green and orange shaded areas show vertical lines and the ending of them, respectively. Spacers of two table candidates (shown in red and blue frames) are detected.

Fig. 4: Example for the column boundary detection algorithm.

other character with 0 (*False*). Lines are padded using multiple 1s to the *width* of a file. Figure 4b shows the bitmap representation for the lines in Figure 4a.

After transforming the line content into a bitmap, the algorithm searches for *vertical lines*. Subsequent text lines, where  $bitmap_w = 1$  for the same character position  $w$ , form a vertical line at  $w$ . Consecutive vertical lines are grouped into *spacers*, which are represented as a set of consecutive *indexes*. Each index represents the character positions of a vertical line. Spacers are *significant*, if they contain more than one vertical line ( $|indexes| > 1$ ) and are not leading ( $0 \in indexes$ ) or trailing ( $width - 1 \in indexes$ ). Significant spacers are mandatory for tables. For the first and the second lines, there are three spacers: columns 4-9, 16-18, and 24-33.

**Append discovered tables** with  $start\_table(counter, l)$ : The algorithm identifies a new table if there is at least one vertical line spanning  $P_{mrc}$  (min row count) text lines. The new table is defined by its starting line index and a set of spacers. For our example, we choose  $P_{mrc} = 2$ . Thus, there was no table discovered after processing the first line. However, after proceeding with the second line of the example, multiple vertical lines span the minimum required number of text lines. The first two lines in Figure 4b show a table  $t_0$  that has three spacers shaded in green.

**Update existing tables** with  $update\_table(t, bitmap)$ : While processing subsequent lines, the existing tables are updated based on the continuation of vertical lines. If a subset of vertical lines belonging to a significant spacer is discontinued, the spacer shrinks or is split into smaller ones so that the continued lines are represented. If the vertical line of an insignificant spacer was discontinued, the algorithm removes it from the set of table spacers. The interruption of all indexes of any significant spacer marks the end of the table.

**Algorithm 2:** Column boundaries detection**Input:** File content  $L$ , file  $width$ , white space characters  $WS$ **Output:** Row range of  $tables$ , table  $boundaries$ 


---

```

1  $counter = \{w \rightarrow 0 \mid 0 \leq w < width\}$  // number of consecutive lines for each vertical
   index
2  $tables = []$  // stores tables (indexed by  $t$ ) with their starting/ending line indexes
3  $boundaries = []$  // stores boundaries of tables  $t$ 
4 for  $k \leftarrow 0$  to  $|L|$  do
5    $closed = []$ 
6    $bitmap = transform(l_k, width)$ 
7   if  $\exists char \in l_k : char \notin WS$  then
8     for  $w \leftarrow 0$  to  $width$  do
9       if  $bitmap_w = 1$  then
10         $counter_w = counter_w + 1$ 
11       else
12         $counter_w = 0$ 
13       for  $t \leftarrow 0$  to  $|tables|$  do
14         $\langle closed_t, boundaries_t \rangle = update\_table(t, bitmap)$  //  $closed_t \in \{0, 1\}$ 
15        if  $\exists w \in 0, \dots, width : counter_w = P_{mrc} \vee closed_{|tables|-1} = 1$  then
16         $tables = tables \cup start\_table(counter, l_k)$ 
17 for  $t \leftarrow 0$  to  $|tables|$  do
18    $boundaries = close\_table(t)$ 
19 return  $tables, boundaries$ 

```

---

**Close tables** with  $close\_table(t)$ : If a closed table covers less than  $P_{msr}$  (min significant rows) rows, insignificant spacers are omitted from the final set, which the algorithm uses to compute the column boundaries. If the number of resulting column boundaries exceeds  $P_{mcc}$  (min column count), they are stored along the table lines in  $boundaries$ . The table is finally closed by removing it from the set of running tables. However, if there are still spacers of that table left, the algorithm creates a duplicate of the table. The clone uses the same set of spacers, but without the discontinued ones. The line index  $k - 1$  is used as the start for the cloned table.

After processing the third line in the example, all spacers of  $t_0$  still exist, whereas the first one shrinks, because the values included in the indexes 4-7 in the third line are zero (shaded orange). The algorithm does not find new tables. When reaching the last line, it updates the last spacer of table  $t_0$  by shrinking it into a smaller, insignificant one. Additionally, vertical lines spanning  $P_{mrc}$  rows were found. A new table  $t_1$  is created using the spacers [8, 13], [16, 18], and 25. The last spacer is insignificant as it contains only one index.

In our example, only the table  $t_0$  is left. The set of column boundaries is the complement of the spacer indexes indicated by the red frames, in all indexes. When using  $P_{msr} = 5$ , the insignificant spacer at index 25 is dropped, as the table has only four rows. The remaining two spacers cover the indexes 8-9 and 16-18. Therefore, the column boundaries for  $t_0$  are: [0, 8), [10, 16), and [19, 34). They are assigned to all four table lines.

The algorithm applies the detected parsing instructions to obtain the resulting interpretations and the values of every field for each line. Leading and trailing whitespace are trimmed from all fields.

## 4.2 Field pattern extraction

In the previous steps, `EXTRACTABLE` collected all valid parsing instructions for each line and returned the resulting interpretations for them. The algorithm generates data types for the values in each interpreted line and uses them in the next step to select the optimal interpretation for the line based on the data type consistency of the field values. Here, we explain how the algorithm determines the data type for a given value.

`EXTRACTABLE` uses a set of 15 domain-agnostic regular expressions to detect known data types, covering all types mentioned in [BNS19]. Additionally, we include regular expressions for Boolean values, file paths, expressions in brackets, and hash-like values. The algorithm assigns the index of the first matching expression to the *known* data type (K). If no data type matches the value, the algorithm falls back to detect the atomic data type for the value. If a field value cannot be covered by any known data type, we use a sequence of atomic type components to describe the type of this value. We support three atomic types: *number* (N), *string* (S), and *other* (O). The remaining class *other* matches everything that is neither a number nor a string.

Empty values (E) are ignored when calculating the consistency of tables. Therefore, the appearance of missing values in combination with another data type in a column has no negative impact on the overall consistency. We use a list of values to represent various forms of empty values, including empty string  $\epsilon$ , N/A, NA, NaN, Null, Unknown, and a sequence of more than one question mark, dash, star, or number sign, respectively.

Finally, we define a pattern as a vector of pattern components. A pattern component can be one of String, Number, Known, Empy, or Other. For the input file, `EXTRACTABLE` detects several valid interpretations, each of which is applied to obtain a set of fields for each line. The field pattern extraction step assigns a pattern for the value of every field.

## 4.3 Table candidate generation

With the value pattern for each field, we calculate a consistency score for a set of lines over corresponding fields across these lines. We introduce a score-based approach that exploits this consistency score to build table candidates. Similar to the detection of column boundaries, `EXTRACTABLE` iterates over all lines and builds table candidates on the fly. It groups line interpretations by two criteria: The primary information is the column count  $n$  and the secondary is the parsing instruction *instr*. The algorithm compares the list of represented groups with the set of existing table candidates  $TC$ . A new table candidate  $C$

is started upon the discovery of an unrepresented group. Table candidates are terminated, if they are no longer represented or if the file end has been reached. Terminated table candidates are passed to the final step of `EXTRACTTABLE`.

The algorithm then adds the corresponding interpretations to the table candidates. Before doing so, it checks whether the current line and the lines in a table candidate are compatible with regard to the *consistency score* of corresponding fields across the lines. If the data types are consistent, the interpretations are appended to the table candidate. If the lines are incompatible, the algorithm starts a new table candidate. Based on our observation, we can assume transitivity: If  $l_k$  is consistent with both  $l_{k-1}$  and  $l_{k+1}$ , then  $l_{k-1}$  and  $l_{k+1}$  are also consistent. Therefore, we compare the current line with only the most recent row of the table candidate. The new table candidate might be created twice: once with and once without using the previous block of compatible lines as a header. The row count of potential headers must not exceed  $P_{mhr}$  (max header rows) and the headers should not include any floats.

Given two rows, our data type-based *consistency score* returns a number between 0 (completely inconsistent) and 1 (perfectly consistent). We consider two interpretations to be compatible if the consistency score exceeds the threshold  $P_{mbc}$  (min block compatibility). We use the *pattern consistency* as the primary measure for the consistency score. The *value uniformity* within columns is calculated to compare consistent tables:

$$score : C \rightarrow \begin{cases} -1, & \text{if } |rich| = 0 \\ 0, & \text{if } |cons| < \lfloor \log_2(|rich|) \rfloor \\ \frac{|cons|}{|rich|} * \frac{1}{|rich|} \sum_{col}^{rich} u(col) & \text{otherwise} \end{cases} \quad (1)$$

where *rich* is the non-empty subset of columns in  $C$ , and *cons* is the homogeneous subset of *rich*. A column is homogeneous if its homogeneity score exceeds one of the thresholds  $P_{mbs}$  (min block score) or  $P_{mcs}$  (min column score). We calculate the score using the homogeneity metric proposed in [Gu11], which considers the distribution of different data types within one column. Based on our experiments, we require at least  $\lfloor \log_2(|rich|) \rfloor$  pattern-consistent columns to compute the table's consistency with the third case of Formula (1). Otherwise, the score for that table is 0. The function  $u$  returns the *value uniformity* of a column  $col$ .

To calculate the uniformity of a column, we first generate the patterns for all values therein and group them by pattern. For each group, we calculate the uniformity using the homogeneity metric for each component in the pattern. For example, "ABC1" and "XYZ0.8" are both mapped to the pattern "SN". Therefore, the uniformity for both "S" and "N" are calculated. For *number* components, we compute the homogeneity of both integers and floats based on their respective counts. A similar calculation is performed for *other* components, where the homogeneity of the values is used. For *known* components of the same type and *string* components, we simply assume that all values are homogeneous and return 1. The value uniformity of *empty* values is undefined. Therefore, the score for the "S" and "N" classes in the above example are 1.0 and 0.5, respectively. Then we denote the

uniformity for this pattern by the maximum uniformity score across all components. Finally, the uniformity of the whole column is the weighted average of the pattern uniformity scores, where the weights are the occurrences of the patterns. The final score of a table is the average uniformity of all *rich* columns.

#### 4.4 Table selection

In the final step, `EXTRACTTABLE` selects a subset of table candidates whose line ranges do not overlap. We model the table candidate selection problem as a *shortest path problem*. We first transform the set of table candidates to a multi-edged directed acyclic graph. The set of vertexes  $V$  represents the line indexes of a file. Each table candidate represents one edge, using the first and last line index for the source node  $src$  and the destination node  $dst$ , respectively. The distance for a given table candidate is calculated using the following formula. Lower distances represent larger and more consistent tables.

$$\begin{aligned}
 dist : C \rightarrow & -score(data(C)) \cdot (m_C - h_C)^2 \\
 & -score(header(C)) \cdot (m_C^2 - (m_C - h_C)^2) - 0.0001 \cdot \text{sgn}(h_C)
 \end{aligned}$$

where  $m_C$  is the number of lines in the table and  $h_C$  is the number of header rows therein. The function calculates the consistency scores  $score(header(C))$  and  $score(data(C))$  for the header and data parts, respectively. When comparing tables of the same size and consistency, we favor tables with a header by subtracting a small constant from the consistency score if a header exists. Before mapping table candidates to edges, the algorithm prunes the ones that have fewer than  $P_{mrc}$  rows and  $P_{mcc}$  columns, or have a consistency score of the data part lower than  $P_{mts}$  (min table score).

After the `DAG` has been filled, adjacent vertexes are linked. We connect each vertex pair  $\langle v, v + 1 \rangle$  by an edge with a distance of  $dis = 0$ . Figure 5 shows the graph for the seven table candidates shown in Table 1. The numbers at the edges represent the distances, and the squared boxes are the table candidate indexes.

Tab. 1: Example table candidates. ‘From’ and ‘To’ fields indicate the beginning and the end indexes of a table candidate.  $SH$  and  $SD$  stand for  $score(header(C))$  and  $score(data(C))$ , respectively.

| # | From | To | $h_C$ | $m_C - h_C$ | $SH$ | $SD$ |
|---|------|----|-------|-------------|------|------|
| 1 | 5    | 35 | 0     | 31          | n/a  | 1.0  |
| 2 | 5    | 35 | 0     | 31          | n/a  | 0.8  |
| 3 | 38   | 45 | 0     | 8           | n/a  | 1.0  |
| 4 | 38   | 55 | 0     | 18          | n/a  | 1.0  |
| 5 | 46   | 55 | 0     | 10          | n/a  | 1.0  |
| 6 | 58   | 75 | 0     | 18          | n/a  | 0.9  |
| 7 | 58   | 75 | 2     | 16          | 1.0  | 0.9  |

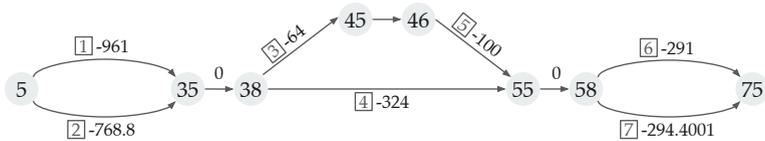


Fig. 5: Table selection graph for example of Table 1

We apply the Bellman-Ford algorithm [Be58] to find the shortest path. In case of a tie, we pick the candidate that has (i) a higher ratio of recognized fields to total fields; (ii) a higher number of *rich* columns; (iii) a lower pattern length; (iv) a lower column count. The best table candidates are found by sorting the edge candidates by their weight and by the criteria above. In the unlikely event that multiple candidates still qualify, we choose the first one.

## 5 Experimental Evaluation

We evaluated `EXTRACTTABLE` on large sets of files taken from open data portals, and compared it with existing solutions regarding accuracy and runtime. The experiments were executed in Python 3.8.5 on a Linux machine. The test system was equipped with an AMD EPYC 7702P CPU with 64 cores, operating at 2 GHz with 512 GB memory.

### 5.1 Datasets

The basis of our ground truth are two existing corpora from related work using plain-text files taken from Mendeley Data, GitHub, and UKdata<sup>5</sup>. The Mendeley data corpus was crawled in August 2020 to study line and cell classification tasks on verbose csv files [JVN21]. This first corpus includes all projects that contain at least one plain-text file and were hosted on Mendeley’s servers. It consists of 235 471 files distributed over 1 554 projects. Within the corpus, we found files of 1 040 different extensions. We kept all files with extensions *.txt*, *.dat*, *.csv*, *.md*, and *.out*, resulting in 94 474 files.

The second corpus was provided as part of [BNS19]. It consists mainly of csv files taken from GitHub and UKdata. A repository hosted on GitHub typically contains a diversity of files required for the development of software. The British government uses UKdata to publish datasets from different departments, such as education, economy, or health. The dataset consists of 5 000 files each from GitHub and UKdata. Using the authors’ script for downloading the corpus<sup>6</sup> from the original sources, some files were no longer available, leaving us with 2 577 and 2 539 files from GitHub and UKdata, respectively.

<sup>5</sup> <https://data.mendeley.com/>, <https://github.com/>, <https://data.gov.uk/>

<sup>6</sup> [https://github.com/alan-turing-institute/CSV\\_Wrangling/](https://github.com/alan-turing-institute/CSV_Wrangling/)

Annotating all almost 100 000 would be too time-consuming, so we selected a subset. We noticed that the Mendeley data source provides a larger variety of files and decided to grant it a larger share in our final dataset. Ultimately, we randomly selected 598 files from Mendeley Data, 176 files from GitHub, and 183 files from UKdata, resulting in 957 files. All files and annotations are publicly available<sup>7</sup>.

We annotated all tables containing at least two columns and two rows. All rows belonging to the same table must have the same column count. Our definition of data tables includes tables with multiple header rows. In our 957 files, we annotated 1 208 tables and obtained first insights into the dataset. A regular table of our ground truth is quite small, with fewer than 1 000 rows and between two and ten columns. Approximately 75% of the 190 ASCII tables have fewer than 100 rows. While files containing a single table are represented using CSV in nine out of ten cases, ASCII tables are used for more than a third of all tables contained in multi-table files. Confirming the general observation of [DMB17], we found that 47% of the CSV tables follow RFC 4180. Also, 1% of the files contained at least one CSV table using a multi-character delimiter, e.g., an arrow (->), multiple slashes (//), or multiple tab or space characters. The majority of fields represent numbers (84%) and only a small portion of cells did not match any of our data types (4%).

## 5.2 Comparison targets

Our comparative analysis regards a simple baseline and four solutions from related work, which we used to evaluate table range selection and parsing accuracy. The simple baseline approach always returns the dialect specified in RFC 4180. By including this baseline when evaluating the parsing results, we were able to gain insights into the complexity of files and the dialect distribution. The *Sniffer* class is part of the *csv* package<sup>8</sup> of Python. Sniffer infers the delimiter by character frequencies across lines. *Hypoparsr* covers multiple parsing steps, such as file encoding detection, dialect detection, and table area detection [DMB17]. While the R package was removed from the Comprehensive R Archive Network by the authors, we used the archived version 0.1.0 from GitHub<sup>9</sup>. Finally, the authors of *CleverCSV* propose a pattern-based approach to infer the dialect of a file [BNS19]. It is capable of handling surrounding text, but does not return the table ranges explicitly. Its command-line tool (version 0.6.7) is available via the Python Package Index<sup>10</sup>.

To evaluate the quality of our table range selection, we used the Python implementation of *Pytheas* [Ch20] published by the authors<sup>11</sup> using the weights that the authors suggest. In addition, we use a *naive* approach for this particular evaluation, which simulates the missing baselines by classifying the complete file content as belonging to a single table.

<sup>7</sup> <https://owncloud.hpi.de/s/uhHJFzC9mNcdF4i>

<sup>8</sup> <https://docs.python.org/3/library/csv.html> (we used Python 3.8.5)

<sup>9</sup> <https://github.com/tdoehmen/hypoparsr>

<sup>10</sup> <https://pypi.org/project/clevercsv/>

<sup>11</sup> <https://github.com/cchristodoulaki/Pytheas/tree/d77b82a>

Other solutions mentioned in related work could not be applied to the table range selection problem. The authors either assumed only a single table to be present within a file, or their implementations did not return the explicit table ranges.

### 5.3 Table range selection

EXTRACTABLE can be configured by a set of ten parameters. Half of the parameters, such as the minimum table dimensions, are subjective and depend on specific tasks. For our datasets, we require tables to have at least two columns and two rows. Based on a related work [Em16], we allow tables to have up to four header rows. The length of a dialect component must not exceed four characters, and all bracket characters are not allowed to appear within the delimiters. To find the optimal values for the remaining five parameters, we ran a grid search on a subset of our ground truth. We found the following settings to be optimal:  $P_{msr} = 4$ ;  $P_{mbc} = 0.71$ ;  $P_{mbs} = 0.31$ ;  $P_{mcs} = 0.51$ ; and  $P_{mts} = 0.51$ .

We measure the quality of the table range selection by calculating the *Intersection over Union* (IOU) for each pair of detected and annotated tables [Re19]. In [Do19] the authors use the IOU metric for evaluating the performance of the table detection in spreadsheets. Since we need to compare only the vertical table boundaries, we use the Jaccard index for the IOU. It returns a number between 0 (no match) and 1 (perfect match).

After calculating the Jaccard index for each table pair, we used the maximum Jaccard index to determine one of four match types: Annotated tables that returned a Jaccard index of 1 for some returned table are a *perfect match*. In contrast, if the maximum Jaccard index of an annotated table is 0, *no match* was found. All remaining annotated tables fall into the category of *partial matches*. We refer to returned tables that have no matching annotated table as *eager match*. Figure 6 shows the match type counts for the naive approach and for Pytheas and EXTRACTABLE (ignoring eager matches). For each individual solution, we excluded those files that were not processed successfully within three minutes. Pytheas finished around 79% of all files, whereas EXTRACTABLE processed around 87% successfully. The naive approach worked on all files due to its nature.

The naive approach returned the correct range for precisely 50% of the tables. The remaining half was classified as a partial match, as every file contains at least one table. Pytheas was able to detect 59% of the table ranges correctly, yet the approach missed every eighth annotated table. The tables that were not recognized are of different sizes and are equally balanced regarding their formats. 6% of all tables returned by Pytheas were not present in the ground truth.

EXTRACTABLE identified the correct table ranges in more than 70% of all tables and missed seven tables (1%). A limitation is the high number of eager matches that are not depicted in the chart. Nearly one out of every six tables returned by EXTRACTABLE is not present in our ground truth, and therefore are false positives. After manually examining a sample of these

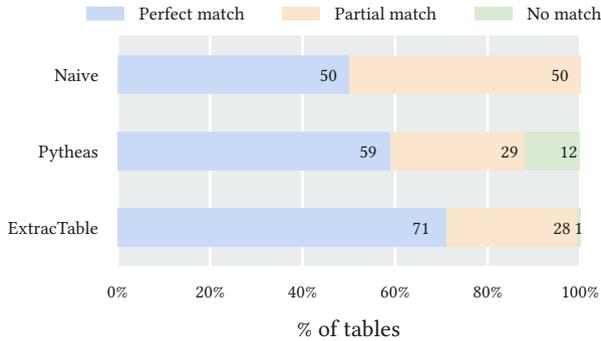


Fig. 6: Table range selection performance (higher number of perfect matches is better).

eagerly matched tables, we realized that it found consistent data tables within unlabeled tabular structures, such as dictionary fragments and single column tables. Pytheas returned fewer false positive tables than `EXTRAC``TABLE`. However, we believe that for users, finding missing tables is more difficult than identifying incorrectly recognized tables in ASCII files, which appear in various shapes and forms. Therefore, the number of eagerly matched tables is a secondary metric compared to the number of correctly matched ones.

## 5.4 Line parsing

To evaluate parsing accuracy, we compared the returned lines of the comparison targets and `EXTRAC``TABLE` line-wise with our annotations. A line was parsed correctly if the returned fields corresponded to the values in the ground truth, taking into account the order. We compared `EXTRAC``TABLE` to four other solutions: RFC 4180, Hypoparsr, Sniffer, and CleverCSV. Some aspects of CSV parsing, such as the handling of space characters in-between fields, are implementation-specific. Therefore, we first extracted the dialects returned by the candidates. We then interpreted lines by feeding the dialect to the same parser. By doing this, we ensured a fair comparison, independent of the parser implementations.

The first experiment examines parsing correctness per table format. Figure 7 shows the ratio of fields that have been correctly parsed for both table formats. For CSV tables, we note that Sniffer, CleverCSV, and `EXTRAC``TABLE` performed similarly well and detected the correct parsing instructions in about 90% of the cases. `EXTRAC``TABLE` achieved slightly lower results than CleverCSV, as it interpreted some tables as ASCII instead of CSV. When disabling the ASCII support, `EXTRAC``TABLE` parsed 94% of all table lines correctly: a higher generality (the ability to also parse ASCII tables) can be a cause for misinterpretations.

`EXTRAC``TABLE` is the only solution optimized for ASCII tables: The remaining solutions recognized merely a small subset of lines correctly. Nevertheless, it is interesting to see that they returned a few correct interpretations. We identified three reasons that led to the

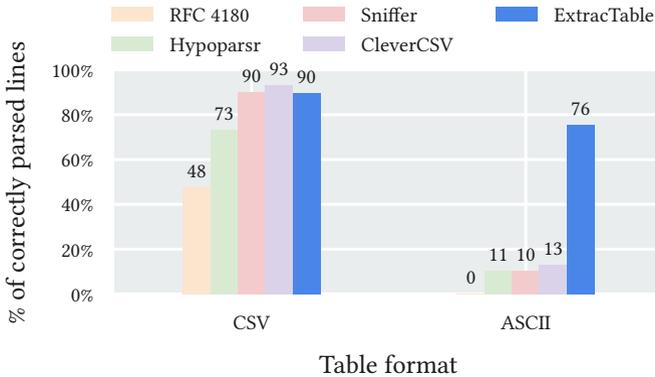


Fig. 7: Parsing accuracy (higher is better).

proper representation of single lines. First, empty lines occurring for a small subset of tables between the header and data part of a table are correct, independent of the used parsing instruction. Second, the nature of ASCII tables lets them use a different number of spaces to separate columns. Solutions besides `EXTRACTABLE` sometimes chose the single space as the delimiter for interpreting these lines. While this does not result in the correct representation of the *whole* table, it sometimes yields the proper interpretations for a *subset* of lines, which is likely to happen for tables with few columns. Third, some tables can be interpreted using both ASCII and CSV. Such a situation may occur if the same number of spaces is used to separate all columns. Independent of these corner cases, we note that `EXTRACTABLE` could correctly interpret 76% of the lines appearing in ASCII tables.

In general, the errors made by `EXTRACTABLE` were independent of the table format but were caused by the table selection, which favors bigger tables. Lines were interpreted incorrectly for three main reasons: (i) over-segmented and under-segmented tables; (ii) short texts surrounding tables; (iii) misinterpretation of tables using tab characters. An annotated table was represented by multiple returned tables that contained partially sorted or similar values (over-segmented). `EXTRACTABLE` under-segmented annotated tables if it found a parsing instruction that could be applied to neighboring tables of the same schema. As the table selection prefers tables with higher row counts, it merges both tables in such cases. Short texts surrounding the tables, such as table titles, causes ASCII tables to merge the first columns as the algorithm tried to include the header row. `EXTRACTABLE` misinterprets CSV tables when it finds a dialect applicable to both the table and the surrounding text. We traced both reasons to our design decision to prefer tables having a higher row count. Finally, CSV tables delimited by the tab character were sometimes misinterpreted as ASCII tables.

To summarize the results, `EXTRACTABLE` performed similarly to `CleverCSV` and `Sniffer` on parsing accuracy for CSV tables. `Hypoparsr` did not perform well, yet it outperformed the RFC 4180 baseline. For ASCII files, only `EXTRACTABLE` could correctly parse a reasonable

number of lines: our approach is more general across the two file types. We assume that the parsing accuracy could be enhanced by pruning non-table lines – a main source of errors.

## 5.5 Runtime

We measured the runtime using Linux’s internal system call `getrusage`. We compared the runtime of our approach to the ones of RFC 4180, `SNIFFER`, `HYPOPARSR`, `CLEVERCSV`, and `PYTHEAS`. To reduce the overall runtime of our experiments, we used a timeout of three minutes, which allowed the slowest approach, `Pytheas`, to finish for more than 70% of the files, covering the majority of the dataset. Only one file fails all approaches with this timeout, which consists of 450 lines, each having 17 365 characters. For each file, we recorded whether the approach was able to process the files within the processing time and returned *some* result. To make the runtime comparable, we kept only files completed by all parsers within the limit. While this could add a bias towards simpler files, it ensures a fair comparison. Figure 8 shows the resulting runtimes per line in milliseconds on a logarithmic scale, based on 551 files.

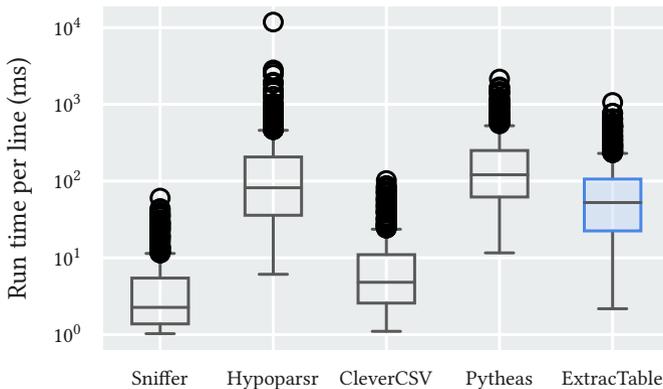


Fig. 8: Runtime comparison using logarithmic scale.

The solution that always returns the configuration of RFC 4180 does not read the file contents and always had a runtime of zero milliseconds. `Sniffer` and `CleverCSV` are both very fast, needing less than 10 ms per line on average. `Hypoparsr`, `Pytheas`, and `EXTRACTABLE` were slower and took 190 ms, 217 ms, and 90 ms, respectively. We acknowledge that all solutions cover a different feature set: While `Sniffer` is a heuristic approach, `CleverCSV` uses a more advanced, pattern-based dialect detection. `Hypoparsr` includes multiple stages, such as encoding detection and normalization. `Pytheas` uses a large set of fuzzy rules to detect table ranges. Our approach includes aspects from different solutions as it covers a wider range of CSV dialects, handles ASCII tables, and is capable of detecting multiple tables within files.

One driver for the longer runtimes is the number of interpretations. This number depends on the chosen configuration, line count, and the actual file content. Lines that are very long or contain many space characters or non-alphanumeric characters take longer to process. The second driver is the number of table candidates. How many table candidates are found depends on the actual content and data type compatibility across lines.

## 6 Summary and Outlook

Tables are stored in arbitrary shapes and forms in plain-text files. To enable automatic information extraction from these types of files, we must first detect the positions of tables and their structures. We proposed the `EXTRACTABLE` algorithm, which tackles the table extraction problem. For a given file, the algorithm first detects and tests possible parsing instructions: dialects and column boundaries for CSV and ASCII tables, respectively. After applying the parsing instructions to the line content, `EXTRACTABLE` infers the data type of each field. It then builds table candidates based on the consistency of data type patterns, field count, and parsing instruction. Finally, the algorithm models the optimal table selection problem as the shortest path problem, and outputs a set of tables for the given file.

To evaluate our algorithm, we annotated a dataset consisting of nearly 1 000 files taken from Mendeley Data, GitHub, and UKdata. We analyzed two aspects of our algorithm: (i) the table range selection; (ii) the parsing accuracy. Our evaluation showed that `EXTRACTABLE` outperforms the other approaches in determining the table ranges, detecting the correct range for more than 70% of the tables. Comparing the parsing results between `EXTRACTABLE` and the related approaches, we found that `CleverCSV` performs best on CSV tables, parsing 93% of the lines correctly. Yet, `EXTRACTABLE` performs similarly well, yielding correct parsing results for 90% of the lines. Our solution was the only one capable of parsing a significant number of ASCII tables and achieved an accuracy of 76%.

While `EXTRACTABLE` supports more complex files, we still had to make a few assumptions, whose relaxation could be interesting future work. This includes the support for cells containing line breaks, as well as spanning rows and spanning columns. The main challenge lies in the scoring of different table candidates. Future work may investigate to what extent the algorithm benefits from learning the structure and content of *typical* tables [VHN22]. We hope that by inferring that knowledge during table selection, wrong interpretations yielding high consistencies can be pruned. Additionally, we identified table selection to be misled by text lines preceding or succeeding a table, because we favor tables with higher row counts. This effect could be reduced by filtering non-table lines as a pre-processing step.

By using the `EXTRACTABLE` algorithm, data scientists can extract tables from a wider variety of plain-text files. Therefore, they spend less time dealing with data wrangling and instead focus on their actual data-driven tasks. While the evaluation returned good results already, we are still far away from handling files fully automatically.

## References

- [An20] Anaconda: 2020 State of Data Science, tech. rep., 2020, URL: <https://know.anaconda.com/rs/387-XNW-688/images/Anaconda-SODS-Report-2020-Final.pdf>.
- [Be58] Bellman, R.: On a routing problem. *Quart. Appl. Math.* 16/, pp. 87–90, 1958.
- [BNS19] van den Burg, G. J.; Nazábal, A.; Sutton, C.: Wrangling messy CSV files by detecting row and type patterns. *Data Mining and Knowledge Discovery* 33/6, pp. 1799–1820, 2019.
- [BTH16] Brickley, D.; Tennison, J.; Herman, I.: CSV on the Web Working Group @ [www.w3.org](http://www.w3.org), tech. rep., 2016, URL: <http://www.w3.org/>, visited on: 04/23/2021.
- [Ch14] Chessell, M.; Scheepers, F.; Nguyen, N.; van Kessel, R.; van der Starre, R.: *Governing and Managing Big Data for Analytics and Decision Makers. IBM Redguides for Business Leaders*, p. 28, 2014, ISSN: 0306-0012.
- [Ch15] Chu, X.; He, Y.; Chakrabarti, K.; Ganjam, K.: Tegra: Table extraction by global record alignment. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*. Pp. 1713–1728, 2015.
- [Ch20] Christodoulakis, C.; Munson, E. B.; Gabel, M.; Brown, A. D.; Miller, R. J.: Pytheas: Pattern-Based Table Discovery in CSV Files. *PVLDB* 13/12, pp. 2075–2089, 2020, ISSN: 2150-8097, URL: <https://doi.org/10.14778/3407790.3407810>.
- [DMB17] Döhmen, T.; Mühleisen, H.; Boncz, P.: Multi-Hypothesis CSV Parsing. In: *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*. New York, NY, USA, pp. 1–12, 2017.
- [Do19] Dong, H.; Liu, S.; Han, S.; Fu, Z.; Zhang, D.: TableSense: Spreadsheet table detection with convolutional neural networks. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)*. Pp. 69–76, 2019.
- [Em16] Embley, D. W.; Krishnamoorthy, M. S.; Nagy, G.; Seth, S.: Converting heterogeneous statistical tables on the web to searchable databases. *International Journal on Document Analysis and Recognition (IJ DAR)* 19/2, pp. 119–138, 2016.
- [Gu11] Guo, P. J.; Kandel, S.; Hellerstein, J. M.; Heer, J.: Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts. In: *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST)*. Pp. 65–74, 2011.
- [HN20] Hameed, M.; Naumann, F.: Data Preparation: A Survey of Commercial Tools. *SIGMOD Record* 49/3, pp. 18–29, 2020.

- [Hu99] Hu, J.; Kashi, R. S.; Lopresti, D. P.; Wilfong, G.: Medium-independent table detection. In: Document Recognition and Retrieval VII. Vol. 3967, International Society for Optics and Photonics, pp. 291–302, 1999.
- [IB72] IBM Corporation: IBM FORTRAN Program Products for OS and the CMS Component of VM/370 General Information./, p. 17, 1972.
- [JVN21] Jiang, L.; Vitagliano, G.; Naumann, F.: Structure Detection in Verbose CSV Files. In: Proceedings of the International Conference on Extending Database Technology (EDBT). Pp. 193–204, 2021, ISBN: 9783893180844.
- [Mo18] Mooney, P.: Kaggle Machine Learning & Data Science Survey, 2018, URL: <https://www.kaggle.com/paultimothymooney/2018-kaggle-machine-learning-data-science-survey>, visited on:
- [PC97] Pyreddy, P.; Croft, W. B.: TINTIN: a system for retrieval in text tables. In: Proceedings of the ACM International Conference on Digital Libraries (DL). Pp. 193–200, 1997.
- [Pi03] Pinto, D.; McCallum, A.; Wei, X.; Croft, W. B.: Table extraction using conditional random fields. In: Proceedings of the International Conference on Information retrieval (SIGIR). Pp. 235–242, 2003.
- [Pr16] Press, G.: Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says. Forbes Tech/, pp. 4–5, 2016, URL: <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/>.
- [Re19] Rezatofghi, H.; Tsoi, N.; Gwak, J.; Sadeghian, A.; Reid, I.; Savarese, S.: Generalized intersection over union: A metric and a loss for bounding box regression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Pp. 658–666, 2019.
- [Sh05] Shafranovich, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) Files, RFC 4180, RFC Editor, Aug. 2005, URL: <https://www.rfc-editor.org/rfc/rfc4180.txt>.
- [SJT03] e Silva, A. C.; Jorge, A.; Torgo, L.: Automatic Selection of Table Areas in Documents for Information Extraction. In: Progress in Artificial Intelligence. Berlin, Heidelberg, pp. 460–465, 2003.
- [VHN22] Vitagliano, G.; Hameed, M.; Naumann, F.: Structural Embedding of Data Files with MAGRITTE. In: NeurIPS Table Representation Learning workshop (TRL). 2022.
- [VJN21] Vitagliano, G.; Jiang, L.; Naumann, F.: Detecting Layout Templates in Complex Multiregion Files. PVLDB 15/3, pp. 646–658, 2021.

# Value-specific Weighting for Record-level Encodings in Privacy-Preserving Record Linkage

Florens Rohde,<sup>1</sup> Martin Franke,<sup>1</sup> Victor Christen,<sup>1</sup> Erhard Rahm<sup>1</sup>

**Abstract:** Privacy-preserving record linkage (PPRL) determines records representing the same entity while guaranteeing the privacy of individuals. A common approach is to encode plaintext data of records into Bloom filters that enable efficient calculation of similarities. A crucial step of PPRL is the classification of Bloom filter pairs as match or non-match based on computed similarities. In the context of record linkage, several weighting schemes and classification methods are available. The majority of weighting methods determine and adapt weights by applying the Fellegi&Sunter model for each attribute. In the PPRL domain, the attributes of a record are encoded in a joint record-level Bloom filter to impede cryptanalysis attacks so that the application of existing attribute-wise weighting approaches is not feasible. We study methods that use attribute-specific weights in record-level encodings and integrate weight adaptation approaches based on individual value frequencies. The experiments on real-world datasets show that frequency-dependent weighting schemes improve the linkage quality as well as the robustness with regard to threshold selection.

**Keywords:** Privacy-preserving record linkage; Bloom filter; Weighting; Value-specific

## 1 Introduction

Record linkage is an essential component in many data integration tasks with multiple data sources. It aims to detect records that belong to the same real-world entity such as a person. Typically, unique record identifiers are not available which would enable a join-like operation [Ch12]. Therefore, records are compared pairwise based on their attributes, such as first name, last name, date of birth and gender. The attribute similarities are used to classify pairs as match or non-match. Often weights are involved in this step to take the different discriminatory power and error rates of attributes into account [WT91]. For example, an equal date of birth is a stronger indicator for a match than an equal gender as there are much more values (and thus each value occurs less often) for date of birth than for gender.

Simple weight-based classification approaches only use attribute-specific weights that are equal for all values of a certain attribute. Thus, the very common last name *Smith* would result in the same weight as the rarer last name *Voigt*. Therefore, the use of value-specific weights based on the frequency of a specific attribute value can increase the linkage quality [WT91]. For uncertain duplicate candidates, e. g., due to a different address as in the following example (see Tab. 1 and 2), the likelihood of a match is higher if the agreeing

---

<sup>1</sup> University of Leipzig & ScaDS.AI Dresden/Leipzig {rohde,franke,christen,rahm}@informatik.uni-leipzig.de

attributes – here first and last name – are rare. This is reflected in a higher record similarity score (weighted average) due to increased weights of those attributes.

Tab. 1: Example of a similarity computation of two records with *common* first and last name.

|            | First name | Last name | Date of birth | ZIP code | City       | Total       |
|------------|------------|-----------|---------------|----------|------------|-------------|
| Record a   | LISA       | SMITH     | 23.09.1973    | 28451    | LELAND     |             |
| Record b   | LISA       | SMITH     | 23.09.1973    | 28075    | HARRISBURG |             |
| Similarity | 1.0        | 1.0       | 1.0           | 0.4      | 0.0        | <b>0.79</b> |
| Weight     | <b>12</b>  | <b>13</b> | 15            | 7        | 7          |             |

Tab. 2: Example of a similarity computation of two records with *rare* first and last name.

|            | First name | Last name | Date of birth | ZIP code | City       | Total       |
|------------|------------|-----------|---------------|----------|------------|-------------|
| Record a   | WYNONA     | VOIGT     | 23.09.1973    | 28451    | LELAND     |             |
| Record b   | WYNONA     | VOIGT     | 23.09.1973    | 28075    | HARRISBURG |             |
| Similarity | 1.0        | 1.0       | 1.0           | 0.4      | 0.0        | <b>0.85</b> |
| Weight     | <b>20</b>  | <b>25</b> | 15            | 7        | 7          |             |

To enable the assignment of globally unique record identifiers multiple data owners share their respective datasets with a trusted institution, called linkage unit, which is responsible for the actual linkage and determines pairs of records considered as a *match*. Using these identifiers the data owners can combine their respective data on matching entities. The exchange of sensitive data, such as identifying personal information, between the data owners or with the linkage unit is, however, restricted by law [CRS20]. Privacy-preserving record linkage (PPRL) addresses this challenge. It has been an active research subject for the last decades [VCV13]. To protect the sensitive data, it is encoded before being sent to the linkage unit which performs the linkage on the encoded data only. A variety of encoding techniques have been proposed, but the most popular and quasi-standard is based on Bloom filters [Gk21]. However, the initially proposed attribute-level encoding [SBR09], where each attribute is encoded in a separate Bloom filter, has been shown to be susceptible to frequency and pattern mining attacks [Vi22]. Therefore, state-of-the-art techniques combine multiple or all attributes into a joint record-level encoding to impede those attacks.

In general, Bloom filter based encodings (both attribute-level and record-level) allow for weighting attributes. Attribute-level encodings are very similar to traditional (plaintext) record linkage with regard to weighting. The attribute similarities can be aggregated to a record similarity, for example, by using a weighted average. The only difference effectively is the use of a similarity function that is suited for the encoded data structure. When using record-level encodings, the data owners can use different parameters per attribute to change the attributes' relative weight in the joint Bloom filter. However, weight adaptation and application in the PPRL context with record-level encodings differ from traditional record linkage as they must be applied by the data owners.

Specifically, we make the following contributions:

- We study the challenges that arise when applying value-specific weighting in the PPRL context to record-level encodings, e. g., the handling of name variations and missing values during the encoding phase.
- We modify record-level encoding techniques for PPRL to allow for frequency-dependent weight adaptation.
- We thoroughly evaluate these techniques and compare them to existing weighting approaches on attribute-level and record-level encodings. Moreover, we analyze the effects of using limited information on value frequencies as the complete information is considered sensitive in the PPRL context.

The paper is structured as follows. In the next section, we discuss Related Work. In Sect. 3 we describe the PPRL encoding and matching process. Then, we discuss weighting-based classification approaches in the PPRL context (Sect. 4) and present an extensive comparative evaluation of the different approaches using a real-world dataset (Sect. 5). Finally, we conclude our work in Sect. 6.

## 2 Related Work

The idea of assigning weights to different attributes when used for calculating similarities between records is part of the probabilistic record linkage approach proposed by Fellegi and Sunter in [FS69]. The weighting of attributes addresses the fact that each attribute has a different number of (possible) values and these values follow a certain distribution. Attributes can also be erroneous or out of date, with some attributes being affected more often than others. Consequently, for each attribute  $i$  two probabilities, namely the  $m$ - and  $u$ -probability, are determined as

$$m_i = P(a_i = b_i, a \in A, b \in B | a \equiv b)$$

$$u_i = P(a_i = b_i, a \in A, b \in B | a \not\equiv b)$$

where  $a$  is a record from database  $A$ ,  $b$  is a record from database  $B$  and  $a_i$  and  $b_i$  are the values of attribute  $i$  of record  $a$  and  $b$ , respectively. With  $\equiv$  we denote the equivalence relation, i. e., both records refer to the same entity. The  $m$ -probability specifies the probability that two records have the same value for attribute  $i$ , given the records refer to the same entity. Ideally,  $m_i = 1$  if all true matches agree on attribute  $i$ . This is exactly the case if attribute  $i$  does not contain any errors. If, for example, 20% of the duplicates have a non-equal value, for instance due to a typographical error, then  $m = 0.8$ . In contrast, the  $u$ -probability specifies the probability that two records have the same value for attribute  $i$  given the records refer to different entities. The  $u$ -probability is low if the attribute has a

wide range of possible values. In contrast, if, for example, an attribute has only two possible and equally likely values, then  $u = 0.5$  as the chance that the attribute agrees for two random records is 50%.  $u$  is typically frequency-dependent as a random agreement is more likely for common than for rare values.

Using the  $m$ - and  $u$ -probabilities the weight  $w_i$  for attribute  $i$  is calculated as

$$w_i = \begin{cases} w_m = \log_2 \left( \frac{m_i}{u_i} \right) & \text{if } a_i = b_i \\ w_u = \log_2 \left( \frac{1-m_i}{1-u_i} \right) & \text{if } a_i \neq b_i \end{cases} \quad (1)$$

The probabilistic record linkage approach by Fellegi and Sunter is the basis for many record linkage approaches and is still frequently used and adapted [Ch12; HSW07].

Herzog et al. [HSW07] propose a method to adjust match and non-match weights also based on the frequency of individual attribute values. Consequently, an attribute-specific and a value-specific weight is used. The authors provide a detailed discussion about the calculation of these weights. Similarly, Zhu et al. [Zh09] propose a scaling factor that is applied directly to the attribute weights of the Fellegi-Sunter approach. The scaling factor is calculated based on the present dataset without an external source of (name) frequencies.

Attribute weighting has been used in the PPRL domain as well. The record linkage and pseudonymization service Mainzliste, which supports Bloom filter based matching, only uses the agreement weights to combine attribute similarity scores to a record similarity using the weighted average [Ro21]. In [Br17], weights are estimated based on partial agreement models for each individual attribute of a sensitive dataset. However, this approach can only be utilized for attribute-level encodings. Ranbaduge et al. proposed decay weights for record-level encodings based on time distances [RC18]. Value-specific weighting approaches, however, have received limited attention so far in PPRL. Giersiepen et al. apply the Fellegi-Sunter approach with frequency-dependent  $u$ -probabilities to encrypted attribute-level hashes [Gi10]. This approach is the standard procedure used by German cancer registries. To the best of our knowledge, no prior work has studied value-specific weight adaptation based on individual value frequencies for record-level encodings so far.

### 3 Background

The general privacy-preserving record linkage process is shown in Fig. 1. We follow a three-party protocol that uses a semi-trusted third party, called linkage unit (LU), to conduct the linkage [CRS20]. The protocol is based on an Honest-But-Curious adversary model which means that all parties follow the protocol but try to learn as much as possible about the sensitive data of others. To protect the privacy of individuals, the quasi-identifying attributes, such as names, dates of birth or addresses, are encoded by the data owners (DO). Often, a preprocessing step is performed before to reduce data quality problems and to

convert the data into a standardized format. Only the encoded quasi-identifiers are then shared with the LU. The LU compares records pairwise and classifies them as *Match* or *Non-Match*. The following subsections explain the matching and encoding phases in more detail.

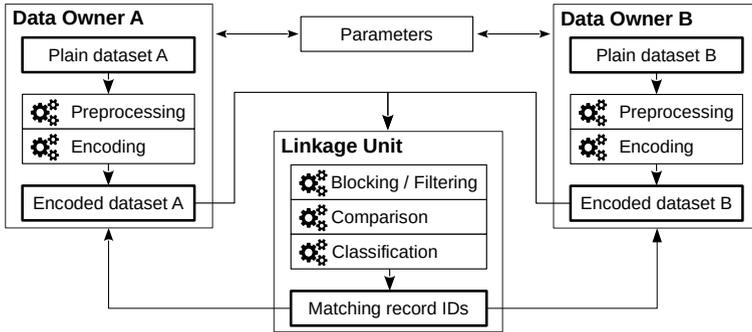


Fig. 1: Privacy-preserving record linkage protocol with two data owners and a semi-trusted third party as the linkage unit.

### 3.1 Encoding

In the encoding phase the plaintext is transformed into an encoded representation that cannot be reverted to its original form. An obvious solution is the use of cryptographic hash functions. However, simple hashes are only suitable for exact matching, as even small differences in the input result in very distinct hash values. Therefore, similarity-preserving encodings have been developed to enable approximate matching of records containing errors or inconsistencies, such as typos or outdated values.

The use of Bloom filters for PPRL has been proposed by Schnell and colleagues [SBR09]. It became the most popular encoding scheme for PPRL in research as well as in real applications [CRS20]. In general, quasi-identifying attributes are split into  $n$  substrings of length  $q$  ( $q$ -grams) to build a set of record features  $F = \{e_1, \dots, e_n\}$  being represented in a Bloom filter. The original strings can be surrounded by leading and trailing padding characters to ensure that all characters are included in the same number of  $q$ -grams, which has been shown to lead to a higher linkage quality [Fr21]. At first, a bit vector of size  $l$  is initialized with each bit set to zero. Moreover,  $k$  hash functions  $h_1, \dots, h_k$  are defined and used to hash (map) the elements of  $F$  into the bit vector. Therefore, *each* hash function is applied on *each* element of  $F$  and produces a position in the range  $[0, l - 1]$  as output. Finally, the bits at the resulting positions are set to one. Given that identical  $q$ -grams are mapped to the same bit positions, a high overlap of  $q$ -grams leads to similar Bloom filters making them suitable for determining the record similarity.

However, due to the deterministic encoding, frequent patterns in the plaintext values will lead to frequently set bit positions in the encoded data and thus enabling frequency attacks. This is true in particular for *attribute-level Bloom filter (ABF)* where a separate Bloom filter is used for each attribute. Consequently, frequently occurring plaintext attribute values can be aligned with frequently occurring Bloom filters. To hamper such attacks, state-of-the-art encodings combine multiple or all attributes into a joint Record-level encoding [Vi22]. The encoding procedure must not be known to the Linkage Unit because otherwise it could conduct dictionary attacks by encoding possible records, e.g., from a similar public dataset, in the same way and infer the membership of a possible record in the dataset. Therefore the encoding output must depend on secrets that are private to the data owners, e.g., by using keyed hash functions.

### 3.2 Matching

In the matching phase records are compared pairwise and classified as match or non-match. To reduce the quadratic complexity of comparing each record of one source with each record of the other source, blocking or filtering techniques can be used [Ch12]. Records that do not meet specific pre-defined blocking or filtering criteria are considered a non-match and thus, are not further compared. Possible blocking keys on plaintext are, for example, year of birth, geographical data items or phonetic encodings of the name. Blocking techniques for Bloom filter based PPRL using Locality-sensitive hashing have been proposed and evaluated [FSR18].

Similarities of Bloom filter encodings can be computed with set similarity measures. In this work we use the Dice coefficient [Di45] which is defined as  $D(a, b) = (2 \cdot |a \cap b|) / (|a| + |b|)$  for Bloom filters  $a$  and  $b$  where  $\cap$  denotes the intersection (logical AND) operation and  $|\cdot|$  the hamming weight of a Bloom filter (number of 1-bits). The resulting similarity score is normalized in the range  $[0, 1]$ . When using ABF encodings, the attribute similarity scores have to be aggregated to a record similarity score, for instance, by computing a weighted average (see Sec. 4.1). If the record similarity score is above a predefined threshold  $t$ , the record pair is classified as a match, otherwise as a non-match.

## 4 Methods

In this section we describe how attribute weights can be applied in the PPRL context, followed by a discussion of methods to adapt the weights depending on the attribute values and their frequencies. Furthermore, we describe approaches to estimate weights and the limitations that arise when transferred to the PPRL domain using record-level encodings.

#### 4.1 Weight application

Attribute weights can be applied in different ways in the PPRL process depending on the encoding strategy. If *attribute-level Bloom filter* (ABF) are used, the linkage unit can compare record pairs attribute-wise. In the probabilistic record linkage theory of Fellegi and Sunter [FS69] (positive) agreement and (negative) disagreement weights are assigned to each attribute depending on whether they are equal or not (see Equation (1)) The total weight is calculated by adding up the respective weights of all attribute pairs. However, this approach does not make use of approximate similarity functions.

Another approach is based on normalized attribute similarity scores in the range  $[0, 1]$  [Ro21]. Those are aggregated into a single record similarity with a weighted average as follows

$$\text{sim}_{record} = \frac{\sum_{i=0}^{N-1} w_i \cdot \text{sim}_i}{\sum_{i=0}^{N-1} w_i} \quad (2)$$

where  $N$  is the number of attributes and the index  $i$  represents attribute  $i$ .

These techniques are equivalent to the application of weights on plaintext data in conventional record linkage as they can make use of attribute-level comparisons. Weights can be determined and applied at the linkage unit during the matching phase. When using record-level encodings, however, attribute weights must be incorporated in the encoding phase at the data owners.

*Record-level Bloom filter* (RBF) encodings, proposed by Durham et al. [Du14], use a sampling based approach. Initially, separate (attribute-level) Bloom filters are generated for each attribute. Based on the respective weights a proportional number of bits is sampled from each attribute-level Bloom filter to construct a record-level Bloom filter. Finally, the bits in the RBF are permuted to ensure that an attacker cannot easily reassign bits of the Bloom filters to specific attributes.

Following the *CLK-RBF* approach by Vatsalan et al. [Va14], weights can be reflected in the number of hash functions  $k_i$  that are used for each attribute  $i$ . The more hash functions are used for an attribute, the more bits in the final Bloom filter are set based on that attribute. Consequently, the influence of that attribute on the Bloom filter similarity is stronger. The number of set bit positions related to a certain attribute also depends on the (average) attribute length. Shorter values consist of fewer record features and thereby fewer bits are set. We compute  $k_i$  with the following equation to ensure that the average number of hash functions of each attribute with respect to the total number of hash functions is proportional to the relative weight of this attribute.

$$\frac{k_i \cdot n_i}{k \cdot \sum_{i=0}^{N-1} n_i} = \frac{w_i}{\sum_{i=0}^{N-1} w_i} \quad \rightarrow \quad k_i = \frac{w_i \cdot k \cdot \sum_{i=0}^{N-1} n_i}{n_i \cdot \sum_{i=0}^{N-1} w_i} \quad (3)$$

where  $w_i$  is the weight,  $n_i$  the average number of features, and  $k_i$  the number of hash functions for attribute  $i$ .  $k$  is the reference number of hash functions and determines the average fill rate (amount of 1-bits relative to the length  $l$ ) of the Bloom filters.

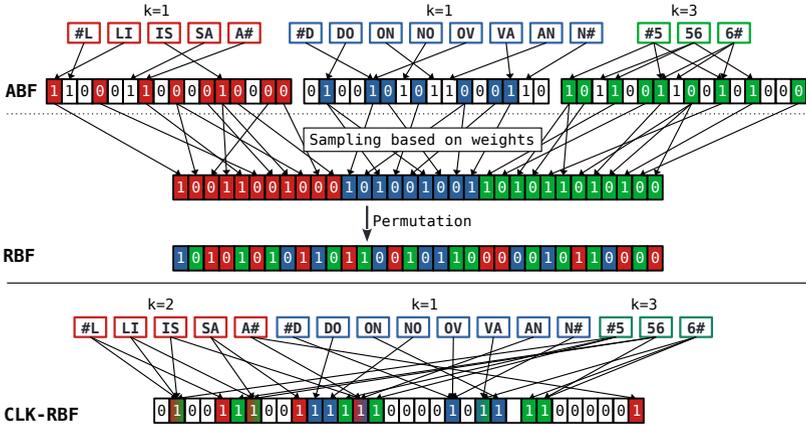


Fig. 2: Encoding of an example record using the weighted record-level techniques RBF and CLK-RBF.

### 4.2 Frequency-dependent weight adaptation

In this section we describe methods to determine value-specific weights that reflect the relative significance of the respective attribute value based on its frequency.

Value frequencies can be incorporated in the Fellegi-Sunter approach by computing a value-dependent  $u$ -probability with  $u_i = f_i/T$ , where  $f_i$  is the absolute frequency of value  $i$  and  $T$  is the total number of values. Consider, for example, an attribute with three possible values 'A', 'B' and 'C' and their respective frequencies in the dataset are 100, 50 and 2 then  $u_A = 100/152 \approx 0.66$  and  $u_C = 2/152 \approx 0.01$ . The likelihood that two random records agree on this attribute is much larger for the most frequent value 'A' than for the rarest value 'C'. Hence, the weights  $w_m$  and  $w_u$  are 0.45 and  $-1.77$  for value 'A' and 6.10 and  $-3.30$  for value 'C' (see Equation (1), assuming a constant  $m = 0.9$ ).

Another approach to modify attribute-level weights is a value-dependent scaling factor  $S$ , so that  $w' = S \cdot w$ . This approach is independent of the method used to determine default weights. Furthermore, it is applicable to other parameters such as the number of hash functions  $k$  in CLK-RBF encodings. We therefore focus on this weight adaptation method.

Zhu et al. [Zh09] proposed a scaling factor defined as

$$S_{\text{Zhu},i} = \sqrt{\frac{T}{Q \cdot f_i}} \tag{4}$$

where  $T$  is the total number of values,  $Q$  is the number of unique values and  $f_i$  is the absolute frequency of the value  $i$ . For values that are more common than the average,  $S_{Zhu}$  is in  $[0, 1)$  and for rare values the factor is larger than 1. In our previous example we would compute  $S_A = \sqrt{152/(3 \cdot 100)} \approx 0.71$  and  $S_C = \sqrt{152/(3 \cdot 2)} \approx 5.03$ . Zhu's scaling factor has, however, two unfavorable properties: (1) The reference for  $S = 1$  is fixed to the mean frequency. Since value frequency distributions, e. g., of names, typically have few very common values and many rare values, this reference can be quite low leading to  $S < 1$  for many mid-common values. (2) The values of the scale factor are biased towards the lower bound. As a consequence, the scale factors are low for values that are not very common.

To address these issues, we propose an alternative scaling factor based on the inverse document frequency (idf) which is defined as  $idf = \log_2(T/f_i)$ . To achieve  $S = 1$  for a desired reference frequency  $f_{ref}$ , we define the scaling factor as

$$S_{idf} = 1 + idf(f_i) - idf(f_{ref}) \tag{5}$$

Moreover, we define  $f_{ref}$  as the frequency of the median attribute value, which is the value in the middle of the ordered list of values with repetition according to the respective frequency. For example, if we have a frequency distribution  $[\{A, 4\}, \{B, 3\}, \{C, 1\}, \{D, 1\}, \{E, 1\}]$ , then the (lower) middle of the list  $[A, A, A, A, \mathbf{B}, B, B, C, D, E]$  is position 5 or value  $B$ . This results in  $S_{idf} < 1$  for values that are more frequent than 3. For  $S_{Zhu}$  the reference (mean) frequency is  $(4 + 3 + 1 + 1 + 1)/5 = 2$ . The median-based approach results in half of the values having a scale factor of below 1 and half above 1.

In practice, the scaling factor  $S_{Zhu}$  can be unreasonably low or high. For instance, in one of the datasets used in our evaluation, we have  $T = 200\,000$  records with  $Q = 20\,060$  unique first names. For the most common name 'James' ( $f_i = 3401$  (1.7 %)) we get  $S_{Zhu} = 0.05$  and for the rarest names with  $f_i = 1$  we get  $S_{Zhu} = 3.2$ . This very large weight reduction for the name 'James' would result in an almost complete disregard of this attribute in the classification which is not desirable.  $S_{idf}$  can even be negative for common values which makes normalization inevitable. Therefore, we normalize and restrict the scales as follows: (1) The scaling factor is normalized to the interval  $[0, 2]$ . We use a separate min-max normalization for scaling factors below and above 1 to ensure that this value is not modified. (2)  $S$  is restricted to a more narrow interval  $[S_{min}, S_{max}]$ , e. g.,  $[0.75, 1.5]$ , to constrain the effect of the weight adaptation.

$$S_{lowest} = \min(S_i) \tag{6}$$

$$S_{highest} = \max(S_i) \tag{7}$$

$$S_{norm,i} = \begin{cases} \frac{S_i - S_{lowest}}{1 - S_{lowest}} & \text{for } S_i < 1 \\ 1 + \frac{S_i - 1}{S_{highest} - 1} & \text{for } S_i \geq 1 \end{cases} \tag{8}$$

$$S_{restricted,i} = \begin{cases} S_{min} + S_{norm,i} \cdot (1 - S_{min}) & \text{for } S_{norm,i} < 1 \\ 1 + (S_{norm,i} - 1) \cdot (S_{max} - 1) & \text{for } S_{norm,i} \geq 1 \end{cases} \tag{9}$$

**ALGORITHM 1:** Computation of value-specific scale factors

---

**Input:**  $AF$ : Lookup table for attribute value frequencies  
 $A_i$ : Attribute value  $i$

**Output:**  $scale_i$ : Scale factor for value  $i$

```

1 if  $A_i$  is in  $AF$  then
2    $freq_i \leftarrow GetFrequency(AF, A_i)$ ;
3    $scale_i \leftarrow ComputeScale(freq_i)$ ;           /* Eq. (4) (Zhu) or Eq. (5) (idf) */
4    $scale_i \leftarrow MinMaxNormalize(scale_i)$ ;     /* Equation (8) */
5    $scale_i \leftarrow RescaleToBoundaries(scale_i)$ ; /* Equation (9) */
6 else
7    $scale_i \leftarrow 1.0$ ;

```

---

The scale factor is applied to weights at the linkage unit for ABF (see Algorithm 2) and at the data owner for RBF (see Algorithm 3) whereas in CLK-RBF the scale factor is applied to the number of hash functions  $k$  (see Algorithm 4). The weight adaptation technique based on a scaling factor requires only a few simple computations as can be seen in Algorithm 1. Therefore, it can be easily integrated into existing frameworks that already support (attribute-level) weighted Bloom filter encodings.

### 4.3 Weight estimation

In this section we describe how weights can be estimated and the issues that arise when applying these methods in the PPRL domain. As described in Sect. 2, a popular method to determine weights is based on the probabilistic approach of Fellegi and Sunter. The computation of the weights requires estimates of the  $m$ - and  $u$ -probabilities for the attributes. Given ground truth data, we can calculate  $m = 1 - e$  where  $e$  is the error rate, i. e., the share of true duplicates with a different value for that attribute. In real-world use cases the error rate must be estimated based on expert and domain knowledge or be determined in a pre-study with a clerical review.

A naive approach to estimate attribute-specific  $u$ -probabilities is  $u = 1/\#uniqueValues$ , which means that the probability that two attribute values agree by chance is equal to the average relative frequency. We use a different approach which is sensitive to the frequency distribution by setting  $u = \sum_{i=0}^{Q-1} p_i^2$ , where  $Q$  is the number of distinct values and  $p_i$  the relative frequency of the  $i$ -th value. As an example, consider the attribute gender that can take the values 'female', 'male' and 'undesignated'. The first two values are nearly equally frequent ( $p = 0.48$ ), the last value, however, is far less common ( $p = 0.04$ ). The estimated probability that the values of two random records agree is  $1/3$  in the naive approach, but 46% when considering the frequency distribution.

---

**ALGORITHM 2:** Linkage with attribute-level similarities and value-specific weighting

---

**Input:**  $\mathbf{R}$ : Dataset with attribute-level encoded records  
 $\mathbf{w}_{default}$ : Default attribute-specific weights  
 $t$ : threshold

**Output:**  $\mathbf{M}$ : Matching record pairs

```

1 Candidates  $\leftarrow$  GenerateRecordPairsWithStandardBlocking( $\mathbf{R}$ );
2  $\mathbf{M} \leftarrow []$ ;
3 for  $Candidate \in$  Candidates do
4    $\mathbf{AP} \leftarrow$  GenerateAttributePairs( $Candidate$ );
5    $\mathbf{AS} \leftarrow$  ComputeAttributeSimilarities( $\mathbf{AP}$ );
6    $\mathbf{w} \leftarrow \mathbf{w}_{default}$ ;
7   for  $sim_i \in$   $\mathbf{AS}$  do
8     if  $sim_i = 1$  then                                /* Adapt only if attributes are equal */
9        $scale \leftarrow$  GetScale( $AP_i$ );                /* Algorithm 1 */
10       $w_i \leftarrow \mathbf{w}_{default,i} \cdot scale$ ;
11    $RS \leftarrow$  ComputeWeightedRecordSimilarity( $\mathbf{w}, \mathbf{AS}$ ); /* Equation (2) */
12   if  $RS > t$  then  $\mathbf{M.append}(Candidate)$ ;

```

---



---

**ALGORITHM 3:** Record-level Bloom filter (RBF) encoding with value-specific weighting

---

**Input:**  $R$ : Plaintext record  
 $\mathbf{w}_{default}$ : Default attribute-specific weights  
 $l_{RBF}, l_{ABF}$ : Length of the record-level / attribute-level Bloom filter  
 $k_{ABF}$ : Attribute-level number of hash functions

**Output:**  $\mathbf{R}_{BF}$ : Encoded Bloom filter record

```

1  $\mathbf{B} \leftarrow []$ ;  $\mathbf{w} \leftarrow \mathbf{w}_{default}$ ;
2 for  $A_i \in R$  do
3    $ABF_i \leftarrow$  GenerateBloomFilter( $A_i, l_{ABF}, k_{ABF}, i$ );
4    $scale \leftarrow$  GetScale( $A_i$ );                        /* Algorithm 1 */
5    $w_i \leftarrow \mathbf{w}_{default,i} \cdot scale$ ;
6  $share \leftarrow$  ComputeProportionalNumbersOfBits( $\mathbf{w}, l_{RBF}$ );
7 for  $ABF_i \in$   $\mathbf{ABF}$  do  $\mathbf{B.append}(SampleBits(ABF_i, share_i))$ ;
8  $\mathbf{R}_{BF} \leftarrow$  Permute( $\mathbf{B}$ );

```

---



---

**ALGORITHM 4:** CLK-RBF encoding with value-specific weighting

---

**Input:**  $R$ : Plaintext record  
 $k_{default}$ : Default attribute-specific number of hash functions  
 $l$ : Length of the record-level Bloom filter

**Output:**  $\mathbf{R}_{BF}$ : Encoded Bloom filter record

```

1  $\mathbf{R}_{BF} =$  InitializeEmptyBloomFilter( $l$ );
2 for  $A_i \in R$  do
3    $scale \leftarrow$  GetScale( $A_i$ );                        /* Algorithm 1 */
4    $k_i \leftarrow k_{default,i} \cdot scale$ ;
5    $\mathbf{BF}_i \leftarrow$  GenerateBloomFilter( $A_i, l, k_i$ );
6    $\mathbf{R}_{BF} \leftarrow \mathbf{R}_{BF} \cup \mathbf{BF}_i$ ;

```

---

The application of this approach in the PPRL domain comes with additional challenges, for attribute-specific weights as well as for frequency-dependent value-specific weights. The estimation of  $u$ -probabilities and the computation of weight scale factors are based on frequency distributions which are not readily available for the datasets to be linked as this information is considered sensitive. We discuss this restriction in Sect. 4.4.

Furthermore, in PPRL with record-level encodings, weights cannot be chosen depending on the agreement/disagreement of attributes. This is because weights must be applied at the data owners where the comparison result is not known yet. Therefore, the two weights of the Fellegi-Sunter model have to be combined into a single weight. Durham et al. [Du14] proposed the range  $w = w_m - w_u$ . The combined weight, however, can be dominated by  $w_m$  if the attributes have a large variety of values which is typically the case for names. As a consequence, we normalize  $w_m$  and  $w_u$  with respect to the maximum/minimum value across all attributes.

$$w_{mi,norm} = \frac{w_{mi}}{\max(w_m)} \quad (10)$$

$$w_{ui,norm} = \frac{w_{ui}}{\min(w_u)} \quad (11)$$

$$w_{i,norm} = \max(w) \cdot (w_{mi,norm} - w_{ui,norm}) \quad (12)$$

#### 4.4 Limited frequency information

Accurate global frequency distributions of attributes across all linked datasets are not readily available in the PPRL context. This information is considered sensitive as it could be used to perform frequency attacks on Bloom filter encodings. In the following, we discuss possible solutions to deal with this limitation.

Frequency distributions can be gathered from an external source on similar datasets, e. g., statistical data from a census of the same geographical region or be computed for the data to be linked. While the first approach is especially useful for smaller datasets where the calculated value counts may not represent the real-world frequency distribution well, the latter ensures that the used frequencies correspond to the actual properties of the dataset.

Each data owner could determine its own source-specific frequency distribution and compute weights based on it. This will result in different weights for identical values. We discuss these effects in Sect. 4.5.

Data owners cannot exchange the complete frequency information as this would leak information on rare values. However, the data owners might be willing and allowed to exchange and combine the relative frequencies of their most common values. While this can increase the linkage quality, it does not affect the privacy as the linkage unit does not learn this information. For  $S_{idf}$  a different reference frequency must be used as the median cannot be determined for an incomplete frequency distribution. We therefore propose using the

least frequent value in the list of most common attribute values as the reference. The scaling factor for this value as well as values not in the list is 1. For values on the list, it is below 1.

The limitation of frequency-dependent weight adjustments to common values will likely lead to smaller effects on the linkage result. Additionally, the weight application can be restricted to certain attributes, e. g. first and last name, because frequency information on other attributes is missing. However, it still could be beneficial with respect to the precision. We experimentally evaluate this effect in Sect. 5.

#### 4.5 Effects of attribute differences in duplicates

Real-world data often contains typographical errors. Besides, names can have natural variations, for instance, the German last name 'Schmidt' with its variants '*Schmid*' or '*Schmitt*'. These varying values occur with different frequencies which leads to different frequency-dependent weights. The consequences of varying weights depend on the weight application technique that is used.

For the RBF approach a single different attribute weight changes the proportions of the weights and hence the sampling rates for all other attribute-level Bloom filters. If these Bloom filters have equal fill rates, the fill rate of the RBF does not change for different weights.

Using CLK-RBF with weight adaption, as described in Algorithm 4, the scaling factor of an attribute  $i$  is applied by changing the number of hash functions  $k_i$  for that attribute only. This does not affect the number of hash functions for the other attributes. Nevertheless, the fill rate of the final Bloom filter is changed as the total number of hash functions is modified.

In Tab. 3, we illustrate the effect of changing a single weight using the two encoding methods. Based on the default weights and the average number of features per attribute, we compute the sampling rates for RBF and the number of hash functions  $k$  for the CLK-RBF approach for an example record  $a$ , that has  $S = 1$  for all attributes. Record  $b$  has the same last name and year of birth, but a different and very common first name. Therefore, we set the scaling factor  $S_b(\text{FN}) = 0.5$ . The sampling rates for all attributes of record  $b$  are changed due to the decreased sum of all weights. For CLK-RBF, however, only  $k_b(\text{FN})$  is adapted. As a consequence, the generated Bloom filter encodings based on the RBF method are more affected by weight variations compared to the CLK-RBF approach.

The same applies if we compute a different scaling factor  $S'_b(\text{FN}) = 0.67$  based on a different frequency information, e. g., when using source-specific frequency distributions (as described in the previous section) where in each distribution the name '*Lisa*' occurs often, but with different relative frequencies. If two sources encode the same record  $b$  based on their respective scaling factors  $S_b$  and  $S'_b$ , the resulting Bloom filters are different. In contrast, using the CLK-RBF approach, this affects only a few hash functions and thus the difference between the Bloom filters is lower.

Tab. 3: Example of a variation of a single weight on record-level encodings ( $l = 1024$ ) based on RBF ( $l_{ABF} = 256$ ) and CLK-RBF ( $k = 20$ ).

|   | First name   | Last name    | Year of birth |
|---|--------------|--------------|---------------|
| <b>Record <math>a</math></b>                                | LISE         | DONOVAN      | 1956          |
| <b>Record <math>b</math></b>                                | LISA         | DONOVAN      | 1956          |
| <b>n (avg. #features)</b>                                   | 7            | 8            | 5             |
| <b><math>w_{\text{default}}</math></b>                      | 12           | 11           | 14            |
| <b><math>S_b</math></b>                                     | 0.5          | 1            | 1             |
| <b><math>S'_b</math></b>                                    | 0.67         | 1            | 1             |
| <b>RBF</b>  |              |              |               |
| <b><math>w_a (= w_{\text{default}})</math></b>              | 12           | 11           | 14            |
| <b><math>w_b</math></b>                                     | 6            | 11           | 14            |
| <b><math>w'_b</math></b>                                    | 8            | 11           | 14            |
| <b>% of sampling for <math>a</math> (<math>w_a</math>)</b>  | 12/37 = 32 % | 11/37 = 30 % | 14/37 = 38 %  |
| <b>% of sampling for <math>b</math> (<math>w_b</math>)</b>  | 6/31 = 19 %  | 11/31 = 36 % | 14/31 = 45 %  |
| <b>% of sampling for <math>b</math> (<math>w'_b</math>)</b> | 8/33 = 24 %  | 11/33 = 33 % | 14/33 = 43 %  |
| <b>CLK-RBF</b>  |              |              |               |
| <b><math>k_a</math> (for <math>S = 1</math>)</b>            | 18           | 15           | 30            |
| <b><math>k_b</math> (for <math>S_b</math>)</b>              | 9            | 15           | 30            |
| <b><math>k'_b</math> (for <math>S'_b</math>)</b>            | 12           | 15           | 30            |

To avoid different weights for attribute variations, such as '*Lisa*' and '*Lise*', we consider the use of frequency distributions based on generalizations of the plaintext values, e. g., using the Soundex phonetic encoding function [OR18]. Consequently, the weight for each value is computed based on the frequency of its generalized value (Soundex code). For example, the Soundex code for both '*Lisa*' and '*Lise*' is 'L200', and thus, the same weights are computed, although the values might have different frequencies.

#### 4.6 Handling missing values

Apart from erroneous attributes, missing values often occur in real-world datasets and a strategy is needed to handle them. When working with plaintext or attribute-level encodings, the linkage unit can detect missing values. Multiple strategies can be used, e. g., ignoring the attribute in the similarity score aggregation or setting its similarity score to 0. When working with record-level encodings the linkage unit cannot detect missing values. If a data owner detects a missing value during the encoding phase the respective weight could be redistributed to the other attributes. However, as the missingness is source-specific, a true match with that value set would be encoded with different weights for all attributes. This would in turn lead to differences in the resulting Bloom filters and thus likely to misclassifications. We therefore do not adapt weights of missing values and simply treat them as empty attributes.

## 5 Evaluation

We evaluate the methods described in the previous section with respect to the linkage quality, while focusing on the following aspects: (1) Quantification of the effects of frequency-dependent weight adaptation. (2) Comparison of weight application approaches in PPRL. (3) Investigation of the effects of limited information on frequency distributions.

### 5.1 Datasets

To study the effects on real-world data, we use a dataset based on the North Carolina Voter Registration (NCVR) database (<https://www.ncsbe.gov/>) provided by Panse et al. [Pa21]. This dataset contains over 120 million historic voter records with person-related attributes such as first name (FN), middle name (MN), last name (LN), year of birth (YOB), place of birth (POB), city, ZIP code and sex. From that dataset we extracted a subset, tagged  $\mathbf{F}$ , by

- (1) Sampling 80 000 individual records (singletons) contained in the snapshot from '2021-01-01' into set  $A_S$  and  $B_S$  each, ensuring that  $A_S \cap B_S = \emptyset$ .
- (2) Sampling 20 000 pairs of records  $a, b$  (duplicates) into sets  $A_D$  and  $B_D$  respectively, where  $a$  is from any snapshot between '2008-01-01' (inclusive) and '2021-01-01' (exclusive), and  $b$  is from snapshot '2021-01-01'. Moreover,  $\forall a, b : (\text{YOB}(a) = \text{YOB}(b)) \wedge \exists \text{attr} \in \{\text{FN}, \text{MN}, \text{LN}, \text{POB}, \text{SEX}\} : \text{attr}(a) \neq \text{attr}(b)$ .
- (3) Constructing the final subsets as  $F_A = A_S \cup A_D$  and  $F_B = B_S \cup B_D$  respectively.

Based on  $\mathbf{F}$  ('Full') we derive another dataset, tagged  $\mathbf{R}$  ('Reduced'), where we removed the attributes middle name and place of birth, thus, making the dataset more challenging to match, see also Tab. 4.

Tab. 4: Description of used datasets.

| Name         | A    | B    | A ∩ B | Attributes                      |
|--------------|------|------|-------|---------------------------------|
| $\mathbf{F}$ | 100k | 100k | 20k   | FN, MN, LN, YOB, POB, CITY, ZIP |
| $\mathbf{R}$ | 100k | 100k | 20k   | FN, LN, YOB, CITY, ZIP          |

For our evaluations with external statistical information we use frequencies of first and last names from the 1990 US Census.<sup>2</sup>

### 5.2 Encoding

We set a fixed length of  $l = 256$  for the attribute-level Bloom filter. The plaintext attributes are preprocessed by removing leading and trailing whitespace, conversion to lowercase

<sup>2</sup> [https://www.census.gov/topics/population/genealogy/data/1990\\_census.html](https://www.census.gov/topics/population/genealogy/data/1990_census.html)

and removal of diacritics before being split into overlapping bigrams using padding. The number of hash functions  $k_i$  is selected based on the average length of each attribute to achieve a unified average fill rate of the respective Bloom filter of approximately 40%. RBF encodings are based on the same ABF parameters. We set the length of the record-level Bloom filter to  $l = 1024$ . For the computation of attribute-specific  $k_i$  in the CLK-RBF encoding we use the reference number of hash function  $k = 12$  (**F**) and  $k = 15$  (**R**) which results in an average Bloom filter fill rate of approximately 40%.

Tab. 5: Attribute properties (availability, average length  $\varnothing l$ ,  $m$ - and  $u$ -probability, normalized weight) and derived encoding parameters for Attribute-level Bloom filter (number of hash functions  $k$ ), Record-Level Bloom filter (share of each attribute) and CLK-RBF (number of hash functions  $k$ .)

| Attr. | Properties |                 |        |        |                   | ABF | RBF   |       | CLK-RBF |      |
|-------|------------|-----------------|--------|--------|-------------------|-----|-------|-------|---------|------|
|       | Avail.     | $\varnothing l$ | m-prob | u-prob | $w_{\text{norm}}$ | k   | %(F)  | %(R)  | k(F)    | k(R) |
| FN    | 99.99 %    | 6               | 0.9300 | 0.0027 | 12.83             | 18  | 19.95 | 24.07 | 15      | 18   |
| MN    | 92.16 %    | 5.1             | 0.4380 | 0.0037 | 7.43              | 21  | 11.55 | –     | 12      | –    |
| LN    | 100 %      | 6.4             | 0.7187 | 0.0010 | 11.14             | 17  | 17.32 | 20.90 | 11      | 15   |
| YOB   | 100 %      | 4               | 0.9900 | 0.0135 | 14.44             | 26  | 22.45 | 27.09 | 24      | 29   |
| CITY  | 99.97 %    | 8.9             | 0.6507 | 0.0193 | 6.63              | 13  | 10.31 | 12.44 | 6       | 7    |
| ZIP   | 99.89 %    | 5               | 0.5318 | 0.0031 | 8.27              | 21  | 12.86 | 15.51 | 11      | 14   |
| POB   | 79.13 %    | 2               | 0.6436 | 0.1513 | 3.57              | 43  | 5.55  | –     | 10      | –    |

### 5.3 Matching

In this work, we focus on the evaluation of comparison and classification rather than techniques to improve scalability. However, to run the experiments in a reasonable time, we use standard blocking to reduce the number of comparisons that need to be computed. To ensure the comparability of the results, we use the same blocking keys independent of the encoding method. For each record we generate blocking keys at the data owners based on the plaintext attribute combinations FN+YOB, LN+YOB and Soundex(FN)+Soundex(LN) and encode them using a cryptographic one-way hash function. These hashed blocking keys are transmitted together with the encoded records to enable blocking at the linkage unit. Additionally, we add a blocking key of the global record id that is unique for each duplicate pair based on the ground truth. This blocking key is used to ensure that no true duplicates are excluded from the comparison.

For attribute-level encodings, attribute pairs with one or both values missing in essential attributes (first name, last name and year of birth) are assigned a similarity score of 0. Other missing attributes are ignored in the weighted average aggregation.

We conduct additional experiments where a post-processing routine on the set of matches is applied, using a symmetric best match strategy (Max1-both) to restrict the result to 1:1 links. In many practical use cases this is reasonable when the sources can be considered duplicate-free and therefore each record of a database has at most one duplicate in the other database [Fr18].

## 5.4 Evaluation measures

We use the standard measures recall, precision and F1-score to evaluate linkage quality. Recall measures the proportion of found true matches from all true matches. Precision measures the proportion of found true matches from all found matches. The F1-score is the harmonic mean of these two measures.

$$\text{Rec.} = \frac{\#\text{TruePos.}}{\#\text{TruePos.} + \#\text{FalseNeg.}}, \quad \text{Prec.} = \frac{\#\text{TruePos.}}{\#\text{TruePos.} + \#\text{FalsePos.}}, \quad \text{F1} = \frac{2 \cdot \text{Rec.} \cdot \text{Prec.}}{\text{Rec.} + \text{Prec.}}$$

We evaluate these measures for similarity thresholds  $t$  in the range of  $[0.7, 1.0]$  in steps of 0.01. In practical record linkage, however, ground truth data is not available and the used thresholds are rarely optimal. For a high linkage quality in real-world applications the results should be stable for a broader range of thresholds. We therefore introduce a loss measure  $L_M^d$  for the linkage quality that describes the maximal loss of measure  $M$  in the threshold range  $[t_{\text{opt}} - d, t_{\text{opt}} + d]$ . Furthermore, we report the area under the curve (AUC) of precision-over-recall as a threshold-independent measure.

## 5.5 Results

The threshold-dependent quality measures are reported for the classification thresholds  $t_{\text{opt}}$  that are optimal for this linkage configuration and used dataset with respect to the F1-score. First, we evaluate different weight adaptation methods based on the scaling factors  $S_{\text{Zhu}}$  and  $S_{\text{idf}}$  (see Tab. 6). We use ABF encodings as described in Algorithm 2 and the full frequency distribution computed for the respective datasets and test multiple scale factor intervals.

Tab. 6: Comparison of weight adaptation methods on Attribute-level Bloom filter.

| DS       | S   | $S_{\text{min}}$ | $S_{\text{max}}$ | AUC          | $t_{\text{opt}}$ | Rec.  | Pre.  | F1    | $\mathbf{L}_{\text{F1}}^{0.01}$ | $\mathbf{L}_{\text{F1}}^{0.03}$ | $\mathbf{L}_{\text{F1}}^{0.05}$ |
|----------|-----|------------------|------------------|--------------|------------------|-------|-------|-------|---------------------------------|---------------------------------|---------------------------------|
| <b>R</b> | –   | –                | –                | 0.841        | 0.85             | 0.786 | 0.787 | 0.787 | 0.029                           | 0.089                           | 0.180                           |
|          | Zhu | 0.5              | 2.0              | 0.869        | 0.82             | 0.813 | 0.813 | 0.813 | 0.032                           | 0.094                           | 0.189                           |
|          |     | 0.5              | 1.5              | 0.867        | 0.82             | 0.810 | 0.812 | 0.811 | 0.031                           | 0.093                           | 0.188                           |
|          |     | 0.75             | 1.5              | 0.856        | 0.84             | 0.786 | 0.817 | 0.801 | 0.022                           | 0.080                           | 0.160                           |
|          | idf | 0.5              | 2.0              | <b>0.884</b> | 0.84             | 0.830 | 0.839 | 0.835 | 0.030                           | 0.091                           | 0.184                           |
|          |     | 0.5              | 1.5              | 0.877        | 0.84             | 0.817 | 0.839 | 0.828 | 0.024                           | 0.083                           | 0.174                           |
|          |     | 0.75             | 1.5              | 0.866        | 0.85             | 0.795 | 0.837 | 0.815 | 0.019                           | 0.087                           | 0.156                           |
| <b>F</b> | –   | –                | –                | 0.918        | 0.83             | 0.809 | 0.914 | 0.859 | 0.008                           | 0.050                           | 0.151                           |
|          | Zhu | 0.5              | 2.0              | 0.933        | 0.79             | 0.853 | 0.904 | 0.878 | 0.019                           | 0.080                           | 0.210                           |
|          |     | 0.5              | 1.5              | 0.932        | 0.79             | 0.849 | 0.904 | 0.875 | 0.017                           | 0.078                           | 0.208                           |
|          |     | 0.75             | 1.5              | 0.927        | 0.81             | 0.834 | 0.906 | 0.868 | 0.014                           | 0.066                           | 0.185                           |
|          | idf | 0.5              | 2.0              | <b>0.940</b> | 0.81             | 0.863 | 0.913 | 0.887 | 0.017                           | 0.072                           | 0.191                           |
|          |     | 0.5              | 1.5              | 0.937        | 0.81             | 0.851 | 0.916 | 0.882 | 0.014                           | 0.064                           | 0.180                           |
|          |     | 0.75             | 1.5              | 0.931        | 0.82             | 0.838 | 0.914 | 0.875 | 0.012                           | 0.058                           | 0.167                           |

All weight adaption configurations improve the linkage quality compared to static weights. However, the  $S_{\text{idf}}$ -based approaches generally show a higher rise of the AUC than the  $S_{\text{Zhu}}$ -based with a maximum improvement by 0.043 (**R**) and 0.022 (**F**) each with a constraint

Tab. 7: Comparison of averaging, attribute-specific and value-specific weighting ( $S_{\text{idf}} [0.5,2]$ ) for different encoding methods.

| DS       | Enc.    | Weighting          | AUC          | $t_{\text{opt}}$ | Rec.  | Pre.  | F1    | $L_{F1}^{0.01}$ | $L_{F1}^{0.03}$ | $L_{F1}^{0.05}$ |
|----------|---------|--------------------|--------------|------------------|-------|-------|-------|-----------------|-----------------|-----------------|
| <b>R</b> | ABF     | –                  | 0.777        | 0.86             | 0.609 | 0.803 | 0.693 | 0.015           | 0.045           | 0.052           |
|          |         | Attribute-specific | 0.841        | 0.85             | 0.786 | 0.787 | 0.787 | 0.029           | 0.089           | 0.180           |
|          |         | Value-specific     | 0.884        | 0.84             | 0.830 | 0.839 | 0.835 | 0.030           | 0.091           | 0.184           |
|          | CLK-RBF | –                  | 0.749        | 0.85             | 0.580 | 0.806 | 0.674 | 0.014           | 0.047           | 0.078           |
|          |         | Attribute-specific | 0.837        | 0.83             | 0.782 | 0.767 | 0.775 | 0.028           | 0.093           | 0.221           |
|          |         | Value-specific     | <b>0.875</b> | 0.81             | 0.804 | 0.849 | 0.826 | 0.015           | 0.070           | 0.189           |
|          | RBF     | –                  | 0.787        | 0.86             | 0.605 | 0.828 | 0.699 | 0.007           | 0.024           | 0.043           |
|          |         | Attribute-specific | 0.845        | 0.85             | 0.770 | 0.806 | 0.788 | 0.015           | 0.074           | 0.179           |
|          |         | Value-specific     | 0.675        | 0.76             | 0.373 | 0.973 | 0.539 | 0.001           | 0.001           | 0.001           |
| <b>F</b> | ABF     | –                  | 0.900        | 0.80             | 0.808 | 0.857 | 0.832 | 0.009           | 0.046           | 0.134           |
|          |         | Attribute-specific | 0.918        | 0.83             | 0.809 | 0.914 | 0.859 | 0.008           | 0.050           | 0.151           |
|          |         | Value-specific     | 0.940        | 0.81             | 0.863 | 0.913 | 0.887 | 0.017           | 0.072           | 0.191           |
|          | CLK-RBF | –                  | 0.845        | 0.77             | 0.714 | 0.804 | 0.756 | 0.009           | 0.034           | 0.067           |
|          |         | Attribute-specific | 0.917        | 0.80             | 0.824 | 0.903 | 0.861 | 0.019           | 0.079           | 0.192           |
|          |         | Value-specific     | <b>0.938</b> | 0.77             | 0.866 | 0.917 | 0.891 | 0.019           | 0.072           | 0.185           |
|          | RBF     | –                  | 0.874        | 0.77             | 0.793 | 0.815 | 0.804 | 0.020           | 0.120           | 0.291           |
|          |         | Attribute-specific | 0.920        | 0.81             | 0.834 | 0.905 | 0.868 | 0.024           | 0.099           | 0.248           |
|          |         | Value-specific     | 0.801        | 0.77             | 0.677 | 0.958 | 0.793 | 0.005           | 0.023           | 0.054           |

interval of  $[0.5, 2.0]$ . The optimal F1-scores show similar increases by 0.048 (**R**) and 0.028 (**F**). Therefore, we use that weight adaption strategy for the following experiments.

We compare the results of the value-specific weight adaptation strategy for the encoding techniques ABF, CLK-RBF and RBF (see Tab. 7). We report two baselines, with and without attribute-specific weights. The latter is implemented by using the arithmetic mean of the attribute similarity scores (ABF), equal number of hash functions  $k$  for all attributes (CLK-RBF) and by sampling equal shares from each attribute (RBF). The value-specific weighting scheme achieves AUC improvements for CLK-RBF comparable to those of the attribute-level application despite the missing restriction of weight adjustments to equal attributes: +0.038 (**R**) and +0.021 (**F**) with respect to the attribute-specific weight baseline and +0.126 (**R**) and +0.093 (**F**) to the averaging baseline. However, with the sampling-based approach (RBF) AUC decreases for value-specific weighting. As we discussed in Sect. 4.5, this is because even a single different weight, e. g., due to a typo, leads to considerably dissimilar Bloom filters. Even with a low threshold of 0.76 the recall is as low as 0.373 (for **R**). Thus, we subsequently focus on the CLK-RBF encoding.

In general, we observe that weight adaptation methods lead to lowered optimal thresholds with increases in recall as well as in precision. While the first is expected when lowering the threshold, the rise of precision suggests that non-match candidates with comparatively high similarity due to common values are less often wrongly classified as matches as these attributes are weighted lower. Moreover, we note that the improved results are also equally or more stable regarding the threshold selection. For **R** with an increase of the F1-score by 0.051,  $L_{F1}^d$  is reduced from 0.028 to 0.015 in  $d = 0.01$  and decreases by 0.023 in  $d = 0.03$ ,

which indicates that a higher linkage quality can be achieved in real-world applications with non-optimal threshold selection (see also Fig. 3).

As explained above, the restriction of the linkage result to 1:1 links is reasonable in some applications and enhances the linkage quality. In order to study whether weight adaption further improves the results, we evaluate the weighting methods for CLK-RBF where the links have been post-processed before the linkage quality assessment (see Tab. 8). The results show increases of AUC, +0.026 (**R**) and +0.011 (**F**), indicating that the weight adjustment technique is beneficial under these conditions as well.

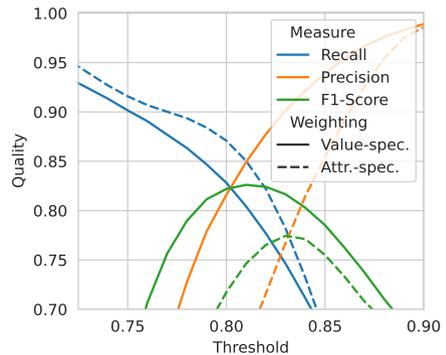


Fig. 3: Comparison of quality measures for attribute- and value-specific weighting on CLK-RBF for **R**.

Tab. 8: Comparison of attribute-specific and value-specific weighting ( $S_{idf} [0.5,2]$ ) for CLK-RBF where the found matches have been restricted to 1:1 links in a postprocessing step (PP).

| DS       | PP  | Weighting          | AUC   | $t_{opt}$ | Rec.  | Pre.  | F1    | $L_{F1}^{0.01}$ | $L_{F1}^{0.03}$ | $L_{F1}^{0.05}$ |
|----------|-----|--------------------|-------|-----------|-------|-------|-------|-----------------|-----------------|-----------------|
| <b>R</b> | yes | Attribute-specific | 0.867 | 0.81      | 0.833 | 0.799 | 0.816 | 0.016           | 0.058           | 0.117           |
|          |     | Value-specific     | 0.893 | 0.80      | 0.820 | 0.876 | 0.847 | 0.007           | 0.038           | 0.093           |
| <b>F</b> |     | Attribute-specific | 0.936 | 0.79      | 0.848 | 0.915 | 0.880 | 0.007           | 0.032           | 0.077           |
|          |     | Value-specific     | 0.947 | 0.76      | 0.881 | 0.927 | 0.904 | 0.012           | 0.045           | 0.101           |

Finally, we study how variations of available frequency information affect the results (see Tab. 9). Again, we report two baselines, with and without value-specific weight adaption, to allow for a comparison with the current state-of-the-art of attribute-specific weights and with value-specific weights under ideal conditions. The results with frequency distributions based on Soundex encodings instead of plaintext values show lower linkage quality, because weights of rare values can be decreased in this setting if these values share the encoding with a common value. Using source-specific frequency distributions the results are almost equal to those with access to the overall frequency information as the distributions are similar. When linking smaller datasets, the distributions will have larger differences, in particular for rare values. We therefore conduct additional experiments where we limit the available frequency information to the most frequent values, as described in Sect. 4.4. The results show that even with a limitation on the 20 most frequent values AUC increases by 0.027 (**R**) and 0.014 (**F**) compared to attribute-specific weights. However, when using external statistical data on the 100 most frequent first and last names only, the linkage quality improvements are comparatively low. The inclusion of information on the frequencies of additional (geographical) attributes could potentially improve the results.

Generally, we see that the quality improvements for **F** are lower than for **R** because the inclusion of information on value frequencies is more relevant in linkage scenarios where fewer attributes are available.

Tab. 9: Comparison of weighting methods with limited frequency information based on CLK-RBF.

| DS | Weighting limitation     | AUC   | $t_{opt}$ | Rec.  | Pre.  | F1    | $L_{F1}^{0.01}$ | $I_{F1}^{0.03}$ | $I_{F1}^{0.05}$ |
|----|--------------------------|-------|-----------|-------|-------|-------|-----------------|-----------------|-----------------|
| R  | Attribute-specific       | 0.837 | 0.83      | 0.782 | 0.767 | 0.775 | 0.028           | 0.093           | 0.221           |
|    | Value-specific           | 0.875 | 0.81      | 0.804 | 0.849 | 0.826 | 0.015           | 0.070           | 0.189           |
|    | ⊢ Soundex-based          | 0.858 | 0.82      | 0.788 | 0.832 | 0.809 | 0.016           | 0.075           | 0.207           |
|    | ⊢ Source-specific        | 0.875 | 0.81      | 0.791 | 0.864 | 0.826 | 0.009           | 0.058           | 0.171           |
|    | ⊢ Top 10                 | 0.858 | 0.83      | 0.789 | 0.836 | 0.811 | 0.014           | 0.077           | 0.218           |
|    | ⊢ Top 20                 | 0.864 | 0.82      | 0.825 | 0.813 | 0.819 | 0.032           | 0.117           | 0.286           |
|    | ⊣ Top 100 Names (Census) | 0.851 | 0.82      | 0.802 | 0.769 | 0.785 | 0.025           | 0.082           | 0.195           |
| F  | Attribute-specific       | 0.917 | 0.80      | 0.824 | 0.903 | 0.861 | 0.019           | 0.079           | 0.192           |
|    | Value-specific           | 0.938 | 0.77      | 0.866 | 0.917 | 0.891 | 0.019           | 0.072           | 0.185           |
|    | ⊢ Soundex-based          | 0.930 | 0.78      | 0.852 | 0.915 | 0.882 | 0.016           | 0.073           | 0.197           |
|    | ⊢ Source-specific        | 0.938 | 0.77      | 0.858 | 0.924 | 0.890 | 0.013           | 0.062           | 0.168           |
|    | ⊢ Top 10                 | 0.928 | 0.79      | 0.845 | 0.913 | 0.877 | 0.018           | 0.080           | 0.212           |
|    | ⊢ Top 20                 | 0.931 | 0.79      | 0.844 | 0.924 | 0.882 | 0.013           | 0.065           | 0.183           |
|    | ⊣ Top 100 Names (Census) | 0.929 | 0.79      | 0.841 | 0.906 | 0.872 | 0.016           | 0.067           | 0.168           |

## 6 Conclusion

Privacy-preserving record linkage enables the integration of sensitive data and thus, its comprehensive analysis. A main challenge is the classification of record pairs as match or non-match based on computed similarities between quasi-identifying attributes of these records. Several studies focus on attribute- and value-specific weighting methods for plaintext data. Nevertheless, only few works adapt these methods in the context of PPRL.

In this work, we apply existing record-level Bloom filter encodings and combine them with frequency-dependent weight adaptation approaches. We extensively evaluate our adapted encoding schemes and compare them with attribute- and record-level Bloom filter encodings. The results show that the modified CLK-RBF encoding outperforms the existing (record-level) methods and achieves comparable results to attribute-level weight application techniques regarding linkage quality and robustness. However, the latter require attribute-level encodings, which are susceptible to cryptanalysis and thus not secure in practical applications. While the weight adaptation disturbs certain frequent bit patterns in the record-level Bloom filters due to the reduced number of hash functions for frequent values, it introduces other frequent patterns in the encoded data as lower weights systematically result in lower fill rates.

In future work, we therefore plan to integrate Bloom filter hardening techniques in our approach to further improve the resistance against cryptanalysis. Furthermore, we will study approaches to estimate *attribute-specific* weights with limited information on frequency distributions and error rates.

**Acknowledgements.** The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany (BMBF) and by the Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus for ScaDS.AI and by the BMBF for the SMITH consortium, grant number 01ZZ1803A.

## References

- [Br17] Brown, A. P.; Randall, S. M.; Ferrante, A. M.; Semmens, J. B.; Boyd, J. H.: Estimating parameters for probabilistic linkage of privacy-preserved datasets. *BMC Medical Research Methodology* 17/95, pp. 1–10, 2017, DOI: 10.1186/s12874-017-0370-0.
- [Ch12] Christen, P.: *Data Matching, Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.
- [CRS20] Christen, P.; Ranbaduge, T.; Schnell, R.: *Linking Sensitive Data, Methods and Techniques for Practical Privacy-Preserving Information Sharing*. Springer, 2020.
- [Di45] Dice, L. R.: Measures of the Amount of Ecologic Association Between Species. *Ecology* 26/3, pp. 297–302, 1945, ISSN: 00129658, DOI: 10.2307/1932409.
- [Du14] Durham, E. A.; Kantarcioglu, M.; Xue, Y.; Toth, C.; Kuzu, M.; Malin, B.: Composite Bloom Filters for Secure Record Linkage. *IEEE Transactions on Knowledge and Data Engineering* 26/12, pp. 2956–2968, Dec. 2014, DOI: 10.1109/TKDE.2013.91.
- [Fr18] Franke, M.; Sehili, Z.; Gladbach, M.; Rahm, E.: Post-processing Methods for High Quality Privacy-Preserving Record Linkage. In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology 2018*. 2018, DOI: 10.1007/978-3-030-00305-0\_19.
- [Fr21] Franke, M.; Sehili, Z.; Rohde, F.; Rahm, E.: Evaluation of Hardening Techniques for Privacy-Preserving Record Linkage. In: *24th International Conference on Extending Database Technology (EDBT)*. Pp. 289–300, 2021, DOI: 10.5441/002/edbt.2021.26.
- [FS69] Fellegi, I. P.; Sunter, A. B.: A Theory for Record Linkage. *Journal of the American Statistical Association* 64/328, pp. 1183–1210, 1969, DOI: 10.1080/01621459.1969.10501049.
- [FSR18] Franke, M.; Sehili, Z.; Rahm, E.: Parallel Privacy-Preserving Record Linkage using LSH-based blocking. *International Conference on Internet of Things, Big Data and Security (IoTBDs)*, 2018, DOI: 10.1007/978-3-030-00305-0\_19.
- [Gi10] Giersiepen, K.; Bachteler, T.; Gramlich, T.; Reiher, J.; Schubert, B.; Novopashenny, I.; Schnell, R.: Performance of record linkage for cancer registry data linked with mammography screening data. *Bundesgesundheitsblatt, Gesundheitsforschung, Gesundheitsschutz* 53/7, pp. 740–747, 2010, ISSN: 1436-9990, DOI: 10.1007/s00103-010-1084-1.
- [Gk21] Gkoulalas-Divanis, A.; Vatsalan, D.; Karapiperis, D.; Kantarcioglu, M.: Modern Privacy-Preserving Record Linkage Techniques: An Overview. *IEEE Transactions on Information Forensics and Security* 16/, pp. 4966–4987, 2021, DOI: 10.1109/TIFS.2021.3114026.

- [HSW07] Herzog, T. N.; Scheuren, F. J.; Winkler, W. E.: *Data Quality and Record Linkage Techniques*. Springer, 2007.
- [OR18] Odell, M.; Russell, R.: The soundex coding system. US Patents 1261167/, 1918.
- [Pa21] Panse, F.; Düjon, A.; Wingerath, W.; Wollmer, B.: Generating Realistic Test Datasets for Duplicate Detection at Scale Using Historical Voter Data. In: *EDBT*. 2021, DOI: 10.5441/002/edbt.2021.67.
- [RC18] Ranbaduge, T.; Christen, P.: Privacy-Preserving Temporal Record Linkage. In: *IEEE International Conference on Data Mining (ICDM)*. Pp. 377–386, 2018, DOI: 10.1109/ICDM.2018.00053.
- [Ro21] Rohde, F.; Franke, M.; Sehili, Z.; Lablans, M.; Rahm, E.: Optimization of the Mainzliste software for fast privacy-preserving record linkage. *Journal of Translational Medicine* 19/33, 2021, DOI: 10.1186/s12967-020-02678-1.
- [SBR09] Schnell, R.; Bachteler, T.; Reiher, J.: Privacy-preserving record linkage using Bloom filters. *BMC Med. Inf. & Decision Making* 9/41, 2009, DOI: 10.1186/1472-6947-9-41.
- [Va14] Vatsalan, D.; Christen, P.; O’Keefe, C. M.; Verykios, V. S.: An Evaluation Framework for Privacy-Preserving Record Linkage. *Journal of Privacy and Confidentiality* 6/1, pp. 35–75, 2014, DOI: 10.1016/j.chemosphere.2016.07.068.
- [VCV13] Vatsalan, D.; Christen, P.; Verykios, V. S.: A Taxonomy of Privacy-Preserving Record Linkage Techniques. *Information Systems* 38/6, pp. 946–969, 2013, DOI: 10.1016/j.is.2012.11.005.
- [Vi22] Vidanage, A.; Ranbaduge, T.; Christen, P.; Schnell, R.: A Taxonomy of Attacks on Privacy-Preserving Record Linkage. *Journal of Privacy and Confidentiality* 12/1, 2022, DOI: 10.29012/jpc.764.
- [WT91] Winkler, W. E.; Thibaudeau, Y.: An application of the Fellegi-Sunter model of Record Linkage to the 1990 U.S. decennial census, Tech. Rep. RR1991/09, Washington, DC: US Bureau of the Census, 1991.
- [Zh09] Zhu, V. J.; Overhage, M. J.; Egg, J.; Downs, S. M.; Grannis, S. J.: An Empiric Modification to the Probabilistic Record Linkage Algorithm Using Frequency-Based Weight Scaling. *Journal of the American Medical Informatics Association* 16/5, pp. 738–745, 2009, DOI: 10.1197/jamia.M3186.

# HYPEX: Hyperparameter Optimization in Time Series Anomaly Detection

Sebastian Schmidl<sup>1</sup> Phillip Wenig<sup>2</sup> Thorsten Papenbrock<sup>3</sup>

## Abstract:

Anomaly detection is a popular activity in time series analytics and covers various techniques for the identification of rare data patterns. These techniques are often presented in the form of automatic anomaly detection algorithms, whose performance depends significantly on the configuration of hyperparameters. Frequently, specifying the hyperparameters of an anomaly detection algorithm manually is particularly difficult because it requires an in-depth understanding of the data and the algorithms' internal behavior. While automatic methods for hyperparameter optimization exist, they require labeled training data and many trials to assess a system's performance before the system can be applied to production data. Hence, existing methods basically shift the efforts from parameter optimization to the labelling of datasets, which is – due to a general lack of high-quality, domain-specific labeled training data – a complex and time-consuming task in time series analytics.

In this paper, we propose a novel hyperparameter optimization framework called HYPEX that learns a parameterization model for anomaly detection algorithms from synthetically generated training data. Based on a (user provided or automatically measured) description of a few time series characteristics, HYPEX quickly suggests effective settings for unseen datasets. The suggestions are based on (i) explainable hyperparameter rules and (ii) learned default parameters, and require no labels for the to-be-analyzed target time series. Our evaluation shows that HYPEX' suggestions significantly improve an algorithm's performance compared to the algorithms' default values and handcrafted heuristics; they often even compete well with the optimal performance achieved with full Bayesian optimization.

**Keywords:** Time Series Anomaly Detection; Bayesian Optimization; Causal Discovery

## 1 The Curse of Hyperparameters

Anomaly detection algorithms for time series data analyze sequences of real-valued, usually time-dependent data for rare subsequence patterns, called anomalies. To obtain good results with these algorithms, various hyperparameters need to be specified. Many of these hyperparameters are algorithm-specific and cover properties, such as *learning rates*, *window sizes*, *maximum cardinalities*, *move distances*, *node degrees*, and *neighbor counts*, some of which hidden behind cryptic names, such as *k*, *phi*, or *delta*. What makes the specification of these hyperparameters hard is that (i) their implications are often hard to guess even by technical experts, (ii) the algorithmic performance is often highly sensitive to the chosen

---

<sup>1</sup> Hasso Plattner Institute, University of Potsdam, Germany, sebastian.schmidl@hpi.de

<sup>2</sup> Hasso Plattner Institute, University of Potsdam, Germany, phillip.wenig@hpi.de

<sup>3</sup> Philipps-Universität Marburg, Germany, papenbrock@informatik.uni-marburg.de

settings, (iii) most hyperparameters are numeric with an infinite parameter space, and (iv) the optimal values often depend on certain characteristics of the input data. For example, many time series anomaly detection algorithms take a *window size* as input. The optimal value for this hyperparameter then correlates, i. a., with the time series' base oscillation frequency, the expected anomaly length, or the time series' extreme values. Most anomaly detection algorithms, therefore, do not perform well with their default parameterization [SWP22].

Hyperparameter optimization is the process of tuning the hyperparameters of an algorithm to a well performing setting [FH19]. This process can be conducted manually or automatically. While the manual search is largely based on domain knowledge, automated approaches find optimal values via, e. g., systematic *Bayesian Optimization* [DC21; DMC16; PGC+99; Sh15] or comprehensive *Grid Search* [Hi12; Le12]. To make any of these approaches work, labeled training data is required to measure the quality of specific hyperparameter settings. Following established machine learning practice, we would collect possibly many labeled time series, and use classical hyperparameter optimization to find optimal settings for some anomaly detection algorithm. The *default values* found with this global optimization will likely perform poorly on a given target dataset because they cannot consider that some hyperparameter values, which are often the most important ones w. r. t. detection accuracy, depend on time series characteristics. Furthermore, real-world data is regularly poorly labeled and, hence, hardly usable for machine learning [SWP22]. For this reason, we would require high-quality training data for every input dataset to effectively optimize an anomaly detection algorithm. However, most practical use cases for time series anomaly detection do not offer any labeled training data, and labeling a sufficient amount of training data is hard.

To ease the hyperparameter optimization process and improve upon globally defined default values, we propose a novel approach: Given a to-be-analyzed target dataset and a to-be-optimized anomaly detection algorithm, we ask the user to specify a set of dataset characteristics, such as *variance*, *min* and *max* values, *oscillation frequencies*, and *expected anomaly lengths*. These characteristics can mostly be profiled automatically from the input data; if profiling is not possible, they are still easier to guess than hyperparameter values and easier to provide than a sufficient amount of labeled training data. Our novel system, then, generates training data based on the provided data characteristics and optimizes the algorithm on this data. The optimization of the hyperparameters is an effective but also expensive process. It, therefore, yields not only a set of possibly robust default parameters, but also a generalizable parameterization model. Because the model learned the relationship between data characteristics and optimal hyperparameter settings, it can quickly optimize the algorithm for different target datasets with different characteristics.

More specifically, we propose *Hyper Parameter Explanation* (HYPEX), a hyperparameter optimization framework that provides hyperparameter optimization models consisting of explainable parameter rules. Assuming that data scientists can specify *discrete* hyperparameters, such as preprocessors, CPU/GPU switches, or ML model types, easily (or at least intuitively), HYPEX focuses on the optimization of *continuous*, numerical hyperparameters. For this, HYPEX uses synthetically generated datasets to determine optimal hyperparam-

| Algorithm           | Family [SWP22] | Float | Integer | Total |
|---------------------|----------------|-------|---------|-------|
| STOMP [Zh16]        | distance       | 1     | 2       | 3     |
| DWT-MLEAD [TKB17]   | distribution   | 1     | 2       | 3     |
| Series2Graph [BP20] | encoding       | -     | 4       | 4     |
| Sub-LOF [Br00]      | distance       | -     | 5       | 5     |
| Donut [Xu18]        | reconstruction | 1     | 5       | 6     |
| Sub-IF [LTZ08]      | trees          | 2     | 3       | 5     |

Tab. 1: Selection of anomaly detection approaches and their hyperparameters by category.

eter values and to identify relationships between (a) different hyperparameters and (b) hyperparameters and dataset characteristics. To learn a reliable hyperparameter model, HYPEX takes as input (i) a time series anomaly detection *algorithm*, (ii) a *hyperparameter configuration* that defines the to-be-optimized algorithm parameters, and (iii) a *dataset generation configuration* that describes possible dataset characteristics. Both configurations require the user to specify value ranges for the hyperparameters and data characteristics, respectively. These ranges should cover the expected use cases and define the scope of the optimization – the larger the ranges, the more general the trained model and the longer the training time. The dataset generation configuration, for example, might specify an oscillation frequency between 1Hz and 2Hz, a maximum value between 0.8 and 1.0, and anomaly lengths between 5s and 30s. Any target dataset in this scope can later be parameterized. Because HYPEX optimizes only numerical hyperparameters, the configurations need to provide settings for all discrete hyperparameters. Once started, HYPEX uses the configurations for automatic training data generation with the GutenTAG [WSP22] dataset generator and for specifying the trials in a systematic *Bayesian Optimization* [PGC+99]. From the many optimal configurations on different datasets, the system then distills all dependencies between hyperparameters and data characteristics into a causal parameter model. With the learned model, we can use HYPEX on any dataset with characteristics in the before specified ranges to propose optimal settings. For this, the user provides the (profiled or estimated) dataset characteristics of a concrete input dataset such that HYPEX can apply them to the parameter rules. Note that all discrete settings, such as type of input data or learning strategy, need to match the settings of the training. With the learned default values and parameter dependencies, HYPEX finally derives well-performing hyperparameter values.

We evaluate HYPEX on the six algorithms shown in Tab. 1 using synthetic datasets in Sect. 4.2. In Sect. 4.4, we exemplarily solve the hyperparameter optimization task for the algorithm Sub-LOF [Br00] on five real-world time series. The algorithms are well-performing representatives from five anomaly detection families that we introduced in a larger evaluative study [SWP22]; they contain between three and six hard-to-optimize numerical hyperparameters of type *float* or *integer*. Our evaluation shows that the automatically suggested hyperparameters improve the anomaly detection quality of the algorithms significantly compared to their default parameters and a manual, heuristics-driven parameterization approach from related work [SWP22]. In summary, HYPEX makes the following contributions:

- (i) **Training data generation:** We extend the GutenTAG data generator with mutation rules to explore numeric dataset characteristics (Sect. 3.1).
- (ii) **Workload distribution:** We design a distributed system for the scalable execution of very many Bayesian optimization tasks (Sect. 3.2).
- (iii) **Causal structure learning:** We propose an algorithm for the identification of dependencies between numeric dataset characteristics and hyperparameters (Sect. 3.3).
- (iv) **Parameter inference:** We discuss the automatic inference of hyperparameters from Bayesian optimization runs on generated data (Sect. 3.4).

## 2 Related Work

Optimizing hyperparameters is a well-known task in many research areas. The three most common approaches for this task are (i) *Random Search* [BB12], (ii) *Grid Search* [Hi12; Le12], and (iii) *Bayesian Optimization* [DMC16; PGC+99; Sh15]. All three algorithms optimize hyperparameters in a way that a user-defined optimization criterion, such as F1 score, AUC-ROC score, or accuracy score, is maximized on a given dataset. While *Random Search* uniformly samples from a given parameter distribution until its time constraint is met, *Grid Search* tests all parameter combinations given by a user-defined parameter grid. *Bayesian Optimization* uses a feedback loop to learn from previous parameter evaluations and improve its parameter suggestions over time. All three algorithms require large amounts of labelled data to evaluate the algorithm or machine learning (ML) model, which is subject to optimization. Our approach overcomes this limitation by generating synthetic, labelled datasets and then transferring learned parameter dependencies. The research community has also come up with systems specifically designed to optimize hyperparameters in the context of anomaly detection. We now discuss two such systems, namely *Opprentice* and *Isudra*.

**Opprentice** *Opprentice* [Li15] is an interesting approach that uses supervised machine learning to improve the quality of anomaly detection in practice. The proposed algorithm focuses on removing the manual work required to adjust parameters and thresholds to reliably detect anomalies. *Opprentice* applies multiple existing anomaly detectors to the incoming data (in parallel) while collecting all detector's outputs, i. e. anomaly scores. Moreover, domain experts are required to label anomalies in the incoming real-world data. The combination of the detectors' outputs and the manually generated labels are used to train a random forest classifier to find reliable detector parameters and thresholds. The authors show that their system removes the manual iterative parameter and threshold tuning. However, domain experts are still required for data labelling purposes.

**Isudra** *Isudra* [DC21] was developed in the context of detecting anomalous data points in clinical health data. The indirect supervision approach for anomaly detection methods

serves to optimize existing unsupervised anomaly detectors and to tune them to concrete application settings. The approach requires clinicians to label sensor time series data with health events. The labelled data gets decomposed into smaller sub-sequences using the sliding window approach with window size  $w_s$ . From each of the resulting windows  $w$ , Isudra extracts descriptive features  $fs(w)$  and, then, applies an anomaly detector  $D$  with a specific set of detector hyperparameters to these features. Once the anomaly detector terminates, the Isudra supervisor calculates a score by comparing the detected anomalies with the ground truth data. Subsequently, Bayesian optimization is used to identify the most effective configuration of window size  $w_s$ , feature set  $fs$ , unsupervised anomaly detector  $D$ , and detector hyperparameters  $\mathcal{H}$ . The authors show that the indirect supervision approach delivers significantly better performance than the alternative methods iForest and One-Class SVM in detecting six out of seven health events. While Isudra automatically optimizes detector parameters, it still requires domain experts, i. e., clinicians, to label a significant amount of anomalous events and, in this way, generate ground truth data.

Our approach also uses the automated hyperparameter optimization technique Bayesian optimization. However, we overcome the requirement of domain experts providing anomaly labels by using a data generator that generates synthetic ground truth data with anomaly labels. Moreover, our work returns a set of parameter-rules, which can be used to adjust a detector's parameters to new, yet unseen datasets.

### 3 Finding Hyperparameter Explanations

In this section, we propose HYPEX, a framework to automatically optimize time series anomaly detector hyperparameters and extract parameter rules without requiring access to manually labelled ground truth data. We use a fully controlled data environment to gain insights on causal dependencies between numerical data characteristics and well-performing parameter sets, as well as relationships between hyperparameters themselves. Because we demonstrate HYPEX in the domain of time series anomaly detection, we first introduce the data (time series) and algorithms (anomaly detection algorithms), we work with.

A *time series* is an ordered sequence  $T = \{T_0, T_1, \dots, T_{n-1}, T_n\}$  of real-valued data points  $T_i \in \mathcal{R}^m$ . An anomaly in such a time series is a subsequence of data points that deviates w. r. t. some measure or model from the frequent patterns in the time series. In our work, we consider w.l.o.g. only univariate time series with a single attribute per data point ( $m = 1$ ). An anomaly detector takes a time series  $T$  as input and computes an anomaly score  $s_i \in \{0, 1\}$  for each data point  $T_i \in T$ . The anomaly score  $s$  indicates the detector's confidence that a data instance  $T_i$  is anomalous. The anomaly scores cannot be compared to the anomaly labels unless they are turned into binary labels using a threshold. Hence, the choice of threshold significantly impacts the anomaly detector's performance and to eliminate it as another tuning parameter, we use the *AUC-PR* score [Br97; DG06] as a performance measure for the anomaly detectors' outputs. This measure is especially popular in applications dealing with learning on imbalanced data, which is the case for anomaly detection.

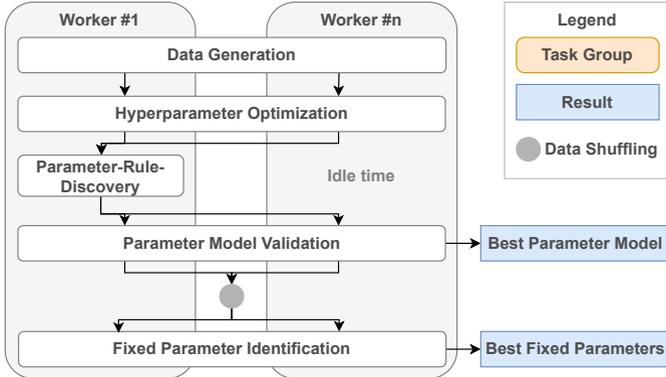


Fig. 1: Control flow of a distributed HYPEX execution consisting of five major steps: (i) data generation (Sect. 3.1), (ii) hyperparameter optimization (Sect. 3.2), (iii) parameter-rule-discovery (Sect. 3.3), (iv) parameter model validation (Sect. 3.3.5), and (v) fixed parameter identification (Sect. 3.4).

HYPEX consists of five consecutive steps that are explained in the following sections: (i) The *data generation* step creates time series, injects anomalies, and labels them (Sect. 3.1), (ii) the *hyperparameter optimization* step uses Bayesian Optimization to find optimal parameters on the generated datasets (Sect. 3.2), (iii) the *parameter-rule-discovery* step finds causal dependencies between data characteristics and the detector’s hyperparameters, which creates a set of parameter model candidates (Sect. 3.3.1 to 3.3.4), (iv) the *parameter model selection* step validates the parameter model candidates to produce the final parameter model that incorporates all significant parameter rules (Sect. 3.3.5), and (v) the *fixed parameter identification* step finds data-independent, fixed parameter values for those parameters that are not covered by the parameter model (Sect. 3.4). The resulting final parameter model and the fixed parameters can be used to calculate future hyperparameters on yet unseen time series based on that time series’ characteristics. It is worth noting that training the parameter model, including data generation, Bayesian optimization and causal inference, is a costly process. For this reason, we propose to parallelize and distribute the efforts. But the training enables HYPEX to afterwards parameterize an algorithm for different input datasets in constant time. Fig. 1 provides a high-level architecture overview of our HYPEX approach. All time-intensive tasks, namely steps (i), (ii), (iv) and (v), are executed on a distributed Dask [Ro15] cluster utilizing multiple parallel worker nodes to speed up the overall runtime.

### 3.1 Data Generation

We use the time series anomaly data generator GutenTAG [WSP22] to generate the synthetic datasets used to optimize the anomaly detector’s parameters. Each generated time series has a base oscillation behavior and potentially multiple injected anomalies of different types. GutenTAG pre-defines six base oscillations and nine anomaly types. When generating a

time series datasets, GutenTAG provides access to the time series, the anomaly labels, and the generation metadata. This information is used to extract and control the dataset characteristics. Fig. 2 shows a generated, univariate, real-valued time series, where the underlying base oscillation simulates electrocardiogram (ECG) data. Two anomalies were added, both indicated by the red background color: the first at position 140 of type *frequency* and length 12, the second at position 240 of type *extremum* and length 1.

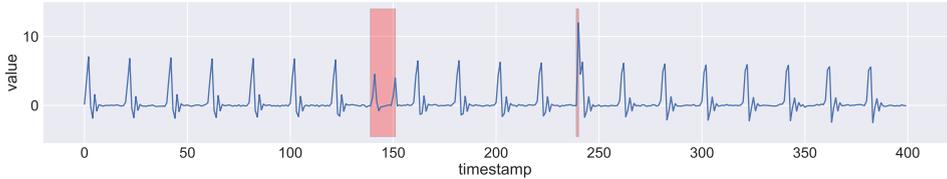


Fig. 2: Exemplary time series generated with GutenTag [WSP22]

To test how an anomaly detector’s parameters need to adapt w. r. t. a change in a specific data characteristic, we introduce *time series mutations*. We define a time series mutation as an attribute change in the GutenTAG configuration used to generate the synthetic time series data. A generated dataset may contain multiple time series mutations, so-called *mutation sets*, which means that two datasets can differ in multiple characteristics. HYPEX uses user-defined mutation sets to generate synthetic datasets that vary in an arbitrary but fixed set of data characteristics, such as anomaly lengths, oscillation frequencies, variances etc. Fig. 3 shows an example of a mutation set that contains a single time series mutation of the attribute *base-oscillation.frequency* applied to the predefined time series *ecg-data*. In this case, HYPEX uniformly samples values for the attribute *base-oscillation.frequency* of the user-provided time series *ecg-data* from the provided range [10, 50]. The sampling creates a total of  $n_{\text{samples}} = 50$  mutated GutenTAG configurations. Subsequently, those 50 different GutenTAG configurations are used to generate the actual 50 datasets. More datasets improve the training quality, but also increase the training time; our experiments use a generously high value of 50 because we focus on quality. By randomly selecting dataset characteristics from the specified ranges, the generated data has all the important characteristics of real data, but with a certain variety and controlled anomalies – for this reason, the learnings on the generated data translate well to real data. To conclude the data generation, HYPEX splits the set of generated time series datasets 60:20:20 into distinct train, validation, and test sets. All physical nodes in the Dask cluster perform the data generation task concurrently.

### 3.2 Hyperparameter Optimization

Once the data generation completes, HYPEX enters the distributed hyperparameter optimization. For each time series dataset in the training split that GutenTAG generated, we independently optimize the given anomaly detector’s parameters such that the algorithm performs well on the selected dataset. Each step, i. e., a particular hyperparameter configuration, of the optimization process is called trial and HYPEX keeps track of the dataset,

---

```

1  name: ecg-data # The time series to mutate
2  n_samples: 50 # The number of mutation sets to generate
3  mutations:
4    - paths: ["base-oscillation.frequency"] # The attribute to mutate
5      dtype: int
6      min: 10
7      max: 50

```

---

Fig. 3: Exemplary mutation of the data characteristic *base-oscillation.frequency* of a user-defined time series called *ecg-data*.

its characteristics, the tested hyperparameters, and the resulting performance scores for all performed trials over all datasets. The number of required trials to achieve reliable results strongly depends on the number and data types of the hyperparameters to optimize.

**Optimization Procedure** HYPEX uses *Bayesian Optimization* for the optimization procedure. While *Random Search* and *Grid Search* simply return the best performing hyperparameters after testing several parameter configurations, *Bayesian Optimization* bases hyperparameter guesses on past parameter evaluations, leading to a faster convergence. Bayesian optimization is commonly used to optimize an objective function  $f$  [Fr18] that is expensive to evaluate, such as tuning an ML model’s architecture, or finding the best hyperparameters for an anomaly detection algorithm. The most widely adopted Bayesian optimization method is Sequential Model-Based Optimization (SMBO). Instead of optimizing the objective function  $f$  directly, SMBO uses a probabilistic model  $P(f(\mathcal{H}) | \mathcal{H})$  as a surrogate for  $f$  [DMC16]. Until a time constraint  $C$  is met, SMBO keeps repeating the following four steps: (i) update the probabilistic model based on previously collected benchmark results, (ii) select the next best guess parameters  $\mathcal{H}$  based on the probabilistic model, (iii) evaluate the objective function  $f$  with the parameters  $\mathcal{H}$ , which is the most expensive step, and (iv) collect and save the benchmark results  $(\mathcal{H}, f(\mathcal{H}))$  for the upcoming optimization steps. The research community came up with several samplers to generate the next best parameter guess based on the previous evaluated parameters [Be11]. Our approach uses the Tree-structured Parzen Estimator (TPE) algorithm. In each iteration, it fits two Gaussian Mixture Models (GMMs) per hyperparameter  $\eta \in \mathcal{H}$ , the first GMM  $l(\eta)$  on the set of well performing hyperparameters  $f(\eta) > y^*$  and the second  $g(\eta)$  on the remaining ones  $f(\eta) \leq y^*$ . The split value  $y^*$  is automatically chosen by the TPE algorithm to match some quantile  $\gamma$  of the observed values for the optimization criterion. In each iteration, for each hyperparameter, TPE chooses the value  $\eta$  that maximizes the ratio  $l(\eta)/g(\eta)$ . Finally, of all the evaluated parameter guesses  $\mathcal{H}$ , the one with the best performance score  $f(\mathcal{H})$  is returned. Our work builds upon the open-source SMBO framework Optuna [Ak19], which implements the TPE algorithm with its *TPESampler*, and we use the *AUC-PR* score [Br97; DG06] as the optimization criterion.

**Task Distribution** HYPEX speeds up the optimization procedure using distributed computing techniques: It splits the workload into smaller sub-tasks, which can be computed independently and in parallel on potentially different physical nodes. For the implementation of this distribution, we have chosen the framework Dask [Ro15]. Dask provides dynamic task scheduling as well as a data collection library - both accessible through a convenient Python API. We wrap each optimization trial into a Dask task such that we can submit the entire set of tasks to the Dask scheduler at once. Dask then controls and schedules the tasks on the cluster's worker nodes. This procedure requires us to ensure that each trial can run on each included physical node, i. e., the synthetic data must be present on all nodes. HYPEX, therefore, seeds the random number generator in GutenTAG such that all physical nodes generate all and the exactly same input datasets. To ensure the Bayesian optimizer bases its parameter suggestion on previous parameter evaluation runs, each trial requires access to not only the trial runs on the same physical cluster node, but to all trials from all cluster nodes. Optuna achieves this trial synchronization by storing trial results in a MySQL database, which HYPEX launches on the Dask scheduler on start up. Each trial first connects to the database to fetch previous trials' results, then suggests a new set of hyperparameters, runs the desired algorithm with the suggested hyperparameters, computes the AUC-PR score from the algorithm's anomaly scores, and finally persists the tested parameters and AUC-PR score in the MySQL database.

### 3.3 Parameter Rule Discovery

We use the optimized hyperparameters found by Optuna (Sect. 3.2) to discover *parameter rules* that specify (a) how hyperparameters depend on other hyperparameters and (b) how hyperparameters depend on specific data characteristics. These data characteristics are represented by the applied time series mutations in our optimization process. To discover the parameter rules, we first present a de-noising step for the trial results, which is the basis of HYPEX's rule discovery (Sect. 3.3.1). Then, we guide through the estimation of an undirected causal graph, the so-called causal skeleton (Sect. 3.3.2). Because the causal skeleton is undirected, we then need to orientate the edges into determinant and dependent hyperparameters/characteristics (Sect. 3.3.3). Once the dependence graph is completed, we discuss how HYPEX compiles the identified parameter rules into a parameter model, which is used later in the process to predict hyperparameters on new, unseen time series dataset based on specific data characteristics (Sect. 3.3.4). Finally, we present an approach to validate discovered parameter rules (Sect. 3.3.5).

#### 3.3.1 Noise Reduction

The set of collected Optuna trials consists of hyperparameter configurations with different performance AUC-PR scores. To find parameter rules, HYPEX should, however, consider only such trails that performed well, because configurations that led to poor trail results

with low AUC-PR scores are not indicative for hyperparameter/characteristic correlations and must, therefore, be considered as noise. To remove a large portion of that noise, we filter out all trials with AUC-PR scores lower than a dynamic threshold  $\gamma$ , which we calculate for each time series in the training split: We define the threshold  $\gamma$  as the top 10% AUC-PR score quantile, which has shown to be a robust selection strategy in all our experiments.

To further improve the explainability of the discovered parameter rules, we also prune parameter rules for hyperparameters with low impact on an anomaly detector's performance. For this, HYPEX uses the parameter importance evaluator fANOVA [HHL14] to identify important hyperparameters. fANOVA trains a random forest regressor to predict the AUC-PR scores based on a given parameter configuration. The random forest regression's feature importance is used to assign importance scores to the anomaly detector's hyperparameters. Then, we prune all hyperparameters with an importance of less than 1% – these parameters, which are usually runtime performance related parameters, do not require optimization.

After noise reduction, HYPEX combines the noise-reduced trials and the information on hyperparameter importance into a matrix  $M$  of dimensionality  $C \times N$ .  $N$  defines the number of trials left after reducing the noise and  $C = p + d + 1$  defines the number  $p$  of anomaly detector hyperparameters with an importance score  $\geq 0.01$  plus the number  $d$  of applied time series mutations and an extra column for each trial's achieved AUC-PR score. Consider an anomaly detector taking two parameters *window size* with an importance score of 99.5% and *random state* with an importance score of 0.5% as input. Moreover, let the datasets contain the single applied time series mutation *base-oscillation frequency*. The resulting matrix  $M$  would consist of three columns, namely *window size*, *base-oscillation frequency*, and the AUC-PR score; *random state* was removed due to its low importance score of 0.5%. The number of rows in  $M$  depends on the performed trials' AUC-PR score distribution. With the assumption of a uniform AUC-PR score distribution and 300 trial runs, we expect  $M$  to have  $0.1 \cdot 300 = 30$  rows.

### 3.3.2 Skeleton Estimation

To identify causal dependencies between columns of the noise-reduced data matrix  $M$  (Sect. 3.3.1), HYPEX performs linear and non-linear independence tests. For these tests, we found that most existing approaches for the detection of non-linear dependencies are too sensitive on our data, given the potentially still tiny signal-to-noise ratio. This is why we use a fairly simple, still powerful, regression-based non-linear independence test with the PC algorithm [Sp00]. The PC algorithm is one of the oldest methods to discover causal dependency graphs [GZS19; Sp00]. It supports plugging in many statistical tests for checking independence and Conditional Independence (CI), which makes it usable in a variety of settings. In HYPEX, we integrate an open-source implementation of the PC algorithm in Python<sup>4</sup> and extend it with own regression-based independence and CI tests

<sup>4</sup> Code available at <https://github.com/keiichishima/pcalg>

called Non-Linear Regression by Transformation (NLRegT) independence test and NLRegT CI test. Note that they are applicable and perform well only on parameters of the numerical data types *integer* and *float*. The PC algorithm uses the following five steps to estimate an undirected version of the true causal graph, which we call the causal graph skeleton:

- (i) Create a fully connected graph  $G$  where each column in the data matrix, i. e., hyperparameter and dataset characteristic, is represented by a node in  $G$ .
- (ii) For each edge  $(A, B) \in G$ , run the (in-)dependency test and remove it from  $G$  if the variables  $A$  and  $B$  are (unconditionally) independent. We use dynamic confidence thresholds  $\alpha$  (for more details, see Sect. 3.3.5).
- (iii) For each of the remaining edges  $(A, B) \in G$  and each set of nodes  $Z = \{Z_1, \dots, Z_n\}$  with  $n \in \mathbb{N}^+$  that are all either connected to  $A$  or  $B$ , remove the edge  $(A, B)$  if  $A$  and  $B$  are conditionally dependent under  $Z$ . Start with  $n = 1$  and repeat this step with increasing set sizes  $n$ . Consider a true causal graph  $A \rightarrow B \rightarrow C$ . Step (ii) finds (unconditional) dependencies between  $\{A, B\}$ ,  $\{B, C\}$ , and  $\{A, C\}$ . However, the dependency  $\{A, C\}$  only gets identified because there exists a path  $(A, B, C)$  between  $A$  and  $C$ . The PC algorithm uses the conditional independence test to identify such dependencies  $\{A, C\}$  and remove them from the estimated causal graph. Note, that it is possible to additionally have an edge  $A \rightarrow C$  in the true causal graph. In this case,  $\{A, C\}$  is not tested conditional dependent and their edge is therefore not removed from the estimated causal graph.

In the following, we explain our NLRegT independence and CI tests, which HYPEX uses with the PC algorithm to estimate the causal graph.

**NLRegT Independence Test** Our NLRegT test is a very robust and powerful independence test for our application scenario. It runs the least squares optimization on pre-transformed data and comes with two transformations by default: *linear* and *hyperbola*. The *linear* transformation in front of the least squares optimization allows testing for relationships between a predicting variable  $u$  and a predicted variable  $v$  of the form  $v = \beta \cdot u + c$ .  $\beta$  being the linear regression's coefficient, and  $c$  being its intercept. The *hyperbola* transformation, on the other hand, enables us to detect causal dependencies between  $u$  and  $v$  of the form  $v = \beta \cdot \frac{1}{u} + c$ . This set of transformations is easily extendable, but our experiments strongly suggested that (at least in the domain of anomaly detection on time series data) *linear* and *hyperbola* dependencies are most common and, hence, sufficient. To fit the non-linear regression with these transformations, our approach takes the following inputs: the slices  $\mathcal{X}$  of the data matrix  $M$  containing the predicting variable and  $\mathcal{Y}$  containing the predicted variable, a data series  $\mathcal{W}$  containing the achieved AUC-PR scores, and a set of transformation functions  $\mathcal{T}$ . For each of the provided transformation functions in  $\mathcal{T}$ , the algorithm transforms the input data of the predicting variable  $\mathcal{X}$  and fits a linear regression on the transformed data  $\mathcal{X}_{\mathcal{T}}$ :

$model \leftarrow fitLinearRegression(X_T, \mathcal{Y}, \mathcal{W}^2)$ . Then, the linear regression of  $\mathcal{Y}$  onto the transformed data  $X_T$  gets fit by using the trials' squared AUC-PR scores  $\mathcal{W}^2$  as sample weights. We thereby increase the weight of trials with higher AUC-PR scores and decrease the weight of those with smaller score values. Afterwards, the fitted regression model itself is scored using the  $R^2$ -Score. Finally, the model with the highest  $R^2$ -Score and the  $R^2$ -Score itself are returned. To test for independence between  $\mathcal{X}$  and  $\mathcal{Y}$ , HYPEX checks whether the calculated  $R^2$ -Score is smaller than a provided threshold  $\alpha$ . Hence, the quality of the causal discovery output strongly depends on the chosen value for  $\alpha$ . We provide more detailed information on how we use  $\alpha$  to generate different model candidates in Sect. 3.3.5.

**NLRegT CI Test** While the provided (unconditional) independence test is generally applicable, our conditional independence test performing  $\mathcal{X} \perp \mathcal{Y} \mid \mathcal{Z}$  assumes the data to be generated under the Additive Noise Model (ANM). For this case, Peters et al. [Pe14] showed that the conditional independence test can be mapped to an unconditional one. The ANM assumes that there is a functional relationship between  $\mathcal{Z}$  and  $\mathcal{X}$  such that  $\mathcal{X} = f(\mathcal{Z}) + \mathcal{N}_x$  with  $\mathcal{N}_x$  being a zero-mean noise independent of  $\mathcal{Z}$ . The same assumption applies to  $\mathcal{Y} = g(\mathcal{Z}) + \mathcal{N}_y$ . These assumptions allow the redefinition of  $\mathcal{X} \perp \mathcal{Y} \mid \mathcal{Z}$  to  $\mathcal{N}_x \perp \mathcal{N}_y$  [Zh17]. Algorithm 1 shows the application of this redefinition: The CI test uses two steps, which are (i) the estimation of the noise terms  $\mathcal{N}_x$  and  $\mathcal{N}_y$ , and (ii) the test for independence between the two estimated noise terms  $\mathcal{N}_x$  and  $\mathcal{N}_y$ . To estimate the two noise terms, we fit the non-linear regression of  $\mathcal{Z}$  onto  $\mathcal{X}$  (Line 2) and  $\mathcal{Y}$  (Line 5); we then calculate the regressions' residuals  $\epsilon_x$  (Line 3) and  $\epsilon_y$  (Line 6). To test for independence between the residuals  $\epsilon_x$  and  $\epsilon_y$ , we once again fit a non-linear regression (Line 8) to check whether the  $R^2$ -Score is larger than a threshold  $\beta$ . HYPEX evaluates the different  $\beta$  threshold values 0.2, 0.4, 0.6, and 0.8. The best performing among these gets selected (for more details, see Sect. 3.3.5).

---

**Algorithm 1** NLRegT CI Test
 

---

```

1: procedure NLREGTCI( $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{W}, \beta$ )
2:    $model_x \leftarrow fitNonLinearRegression(\mathcal{Z}, \mathcal{X}, \mathcal{W})$ 
3:    $\epsilon_x \leftarrow \mathcal{X} - model_x.predict(\mathcal{Z})$ 
4:
5:    $model_y \leftarrow fitNonLinearRegression(\mathcal{Z}, \mathcal{Y}, \mathcal{W})$ 
6:    $\epsilon_y \leftarrow \mathcal{Y} - model_y.predict(\mathcal{Z})$ 
7:
8:    $score \leftarrow fitNonLinearRegression(\epsilon_x, \epsilon_y)$ 
9: return  $score > \beta$ 

```

---

### 3.3.3 Edge Orientation

Until now, we found correlated hyperparameter-hyperparameter and hyperparameter-characteristic edges as undirected relationships. To derive hyperparameters from other hyperparameters or dataset characteristics, these relationships need to be oriented, which is

done in three steps: (i) incorporation of prior knowledge on dataset characteristics, (ii) estimation of the Completed Partially Directed Acyclic Graph (CPDAG), and (iii) conversion of the CPDAG to a Directed Acyclic Graph (DAG).

In step (i), we incorporate a task-specific constraint: Because we aim to find good hyperparameters given certain dataset characteristics, edges need to point from characteristics to parameters. Hence, given the edge  $(A, B)$  with orientation  $A \rightarrow B$ , HYPEX removes all edges  $(A, B)$  from the graph skeleton where the node  $B$  is a dataset characteristic. Step (ii) then executes the PC algorithm’s CPDAG estimation. The PC algorithm is guaranteed to converge to the Markov Equivalence Class (MEC) under the causal Markov condition and faithfulness assumption and when there is no undiscovered confounder [GZS19]. Consider, for example, a true causal graph  $G$  containing only two nodes  $A$  and  $B$  with a single undirected edge  $\{A, B\} \in G$ . Here, the PC algorithm identifies  $A$  and  $B$  as dependent on one another, but it cannot decide the dependency direction. Therefore, the resulting graph contains an undirected edge between the nodes  $A$  and  $B$ . Such a graph that represents the MEC and possibly contains a mixture of directed and undirected edges is called a CPDAG. To estimate the CPDAG, the PC algorithm executes the following two steps [GZS19]:

- (i) Search for *v-structures* and orient edges accordingly. A *v-structure* is a triple of nodes  $(A, B, C)$  such that there exist the undirected edges  $\{A, B\} \in G$  and  $\{B, C\} \in G$ , but  $\{A, C\} \notin G$  and the node  $B$  is not contained in the set  $Z = \{Z_0, \dots, Z_n\}$  under which  $A$  and  $C$  were tested conditionally independent. The edges in such a *v-structure* are oriented  $A \rightarrow B$  and  $C \rightarrow B$ .
- (ii) Use *orientation propagation* to orient possibly many of the remaining edges. To do so, search for a triple of nodes  $(A, B, C)$  such that there exists a directed edge  $(A, B) \in G$ , an undirected edge  $\{B, C\} \in G$ , but no edge between  $A$  and  $C$ . In each of the found triples, the undirected edge  $\{B, C\}$  gets oriented  $B \rightarrow C$ .

The resulting CPDAG might still contain undirected edges, but our final parameter model requires all discovered parameter rules to have a clear orientation. To ensure this requirement, step (iii) of the edge orientation converts the estimated CPDAG to a DAG with a colored depth-first search [Su17; ZG07]. The search removes back edges in the graph, thus breaking existing cycles and creating a DAG with no undirected edges or cyclic dependencies.

### 3.3.4 Parameter Model

Given the dependency DAG, HYPEX now trains a set of *parameter models* that predict optimal values for dependent anomaly detector hyperparameters from the set of data characteristics and other hyperparameters: For each node in the DAG, HYPEX identifies all predecessor nodes and fits the NLRegT model once again using the trials’ squared AUC-PR score as the sample weight. While the NLRegT model previously contained just a single

feature, which was used to predict the target variable, the number of features here depends on the number of predecessors in the graph. The parameter model stores at most one NLRegT model for each anomaly detector hyperparameter. Once the parameter models are trained on generated data, HYPEX can use them to predict hyperparameter values on unseen data. For this, the algorithm iterates over the estimated DAG in topological order. In each step, it uses the stored NLRegT model to predict the respective parameter based on all previously estimated parameters and data characteristics. In Sect. 3.4, we discuss how HYPEX assigns fixed values to the hyperparameters that are not covered by the parameter model.

### 3.3.5 Parameter Model Selection

Our experiments show that the optimal  $\alpha$  and  $\beta$  thresholds to choose for the parameter-rule-discovery (Sect. 3.3.2) are algorithm- and base oscillation-specific. Therefore, we run the parameter-rule-discovery for different  $\alpha$  and  $\beta$  thresholds, leaving us with a set of parameter model candidates. The set of  $\alpha$  thresholds to test is determined by fitting a non-linear regression on each pair of variables in the data matrix  $M$ , rounding the regressions'  $R^2$ -scores to two decimal places and considering only  $R^2$ -scores  $\geq 5\%$  as significant; the set of  $\beta$  thresholds is fixed and set to  $\{0.2, 0.4, 0.6, 0.8\}$ . These settings have been found via systematic ablation tests and showed to be dataset and algorithm independent; hence, we propose them as default settings for HYPEX. For each unique combination of  $\alpha$  and  $\beta$  thresholds, we create a parameter model candidate by running our parameter-rule-discovery using the respective threshold values. Subsequently, we measure the parameter model candidates' performances on the validation data split, which contains 20% of the generated time series datasets. To test a candidate's performance, we use the parameter model to predict the anomaly detector's parameters based on data characteristics. All parameters that are not covered by the parameter model candidate are filled up with uniformly sampled random values. For each pair of model candidate and time series dataset, we independently sample non-covered parameters 10 times and, finally, choose the parameter model with the highest mean AUC-PR score across all conducted tests.

## 3.4 Fixed Parameters

Our parameter models are expected to cover only such anomaly detector hyperparameters that depend on either a data characteristic or another hyperparameter. For the remaining, data-independent anomaly detector hyperparameters, we identify generally well-performing, fixed values. To find these values, HYPEX again uses the Bayesian optimizer to optimize the mean AUC-PR score across all generated validation time series datasets. For each time series, we utilize the parameter model to predict the data-dependent hyperparameters based on the contained time series anomalies. Only the data-independent parameters are subject to optimization in this final step. Eventually, the hyperparameter values of the best performing trial are selected as generally applicable, fixed parameters for the tested anomaly detector.

The final parameter model holds a combination of NLRegT models and fixed parameters, thus being able to predict all anomaly detector parameters on unseen time series data.

## 4 Evaluation

In this section, we evaluate HYPEX on a variety of anomaly detection algorithms and time series datasets. We start with the explanation of the experimental setup (Sect. 4.1), then compare the performance scores achieved by our parameter suggestions with relevant alternative approaches (Sect. 4.2), review HYPEX’s sensitivity to automatically chosen thresholds (Sect. 4.3), and show the application of HYPEX to real-world data (Sect. 4.4).

### 4.1 Experimental Setup

We evaluate our approach<sup>5</sup> on six anomaly detection algorithms and four time series dataset groups containing different base oscillations and anomaly types. The included base oscillation behaviors are (i) sine, (ii) ECG, (iii) random walk, and (iv) cylinder bell funnel [WSP22]. Each base oscillation is considered separately, as we find this to have a large impact on possible parameter rules and algorithm behaviors. All generated time series datasets consist of 10,000 individual data points and contain 3, 6, or 9 same-length anomalies at different positions of types (i) variance, (ii) frequency, or (iii) pattern [WSP22]. We apply time series mutations to (a) the base oscillation frequency (only applicable for sine and ECG), (b) the base oscillation variance, (c) the length of anomalies, and (d) the number of anomalies. For each of the four dataset groups characterized by the four base oscillation behaviors, we incorporate 50 time series mutations. We recall that 20% of the generated datasets are reserved for evaluation purposes only (Sect. 3.1). We evaluate our approach on algorithms from five out of six anomaly detector families defined by Schmidl et al. [SWP22]. The algorithms stem from the areas (i) distance (STOMP [Zh16] and Sub-LOF [Br00]), (ii) distribution (DWT-MLEAD [TKB17]), (iii) encoding (Series2Graph [BP20]), (iv) reconstruction (Donut [Xu18]), and (v) trees (Sub-IF [LTZ08]). Each algorithm has between 3 and 6 hyperparameters that are subject to optimization. For the evaluation, we use the AUC-PR score to measure the algorithms’ detection quality.

We do not have any information on the true relationships between data characteristics and hyperparameters for any of the anomaly detectors, on which we evaluate our framework HYPEX. Therefore, we measure the quality of our parameter model and fixed parameter suggestions by comparing them to (a) the algorithms’ default parameters, (b) the manually tuned parameter recommendations of TimeEval [SWP22], and (c) the optimization results achieved by the Bayesian Optimizer Optuna (*full optimization*). While each detector’s default hyperparameter configuration stays constant, regardless of which time series it runs

---

<sup>5</sup> Code and evaluation scripts: <https://github.com/HPI-Information-Systems/hypex>

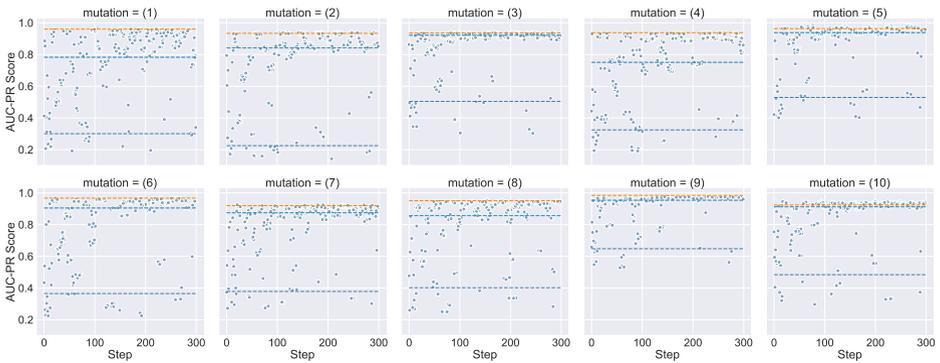


Fig. 4: Evaluation of Donut [Xu18] on 10 sine time series comparing our  $\bullet\bullet$  *Parameter Model* with Donut's  $\square\square$  *Default Parameters*, the  $\square\square$  *Timeeval Heuristics*, and a  $\bullet$  *Full Optimization* run.

on, HYPEX's parameter model as well as TimeEval's parameter suggestions use specific data characteristics to adapt a selected subset of the hyperparameters to the time series input while keeping others constant. The *full optimization*, however, tunes every single parameter to the specific input time series. Thus, the best scores of the full optimization represent an upper bound for any optimization effort (which is achievable only with suitable training data) while the algorithm's default parameters represent a lower bound for HYPEX's performance; the TimeEval results show what results can be expected with significant manual effort.

## 4.2 Parameter Model Performance

In this section, we first show a single anomaly detector's performance using HYPEX's suggested hyperparameters. Then, we present a general overview of the performance achieved on the tested algorithms and base oscillations.

Fig. 4 visualizes our evaluation results on the anomaly detector Donut [Xu18] and 10 time series datasets with base oscillation *sine* that cover various time series mutations with different dataset characteristics and, in particular, different types and numbers of anomalies in each time series. It compares the performance of  $\blacksquare$  HYPEX's parameter model with the performance achieved by (i)  $\blacksquare$  the detector's default parameters, (ii)  $\blacksquare$  TimeEval's manually found default parameters and heuristics, and (iii)  $\blacksquare$  a full Bayesian optimization run (*full optimization*). While the default parameters, the TimeEval heuristics, and our parameter model recommend a single hyperparameter configuration per input dataset, the full optimization is granted 300 trials to optimize Donut's hyperparameters for each of the 10 time series. Our first insight is that especially Donut's default parameters perform poorly, which clearly emphasizes the need for hyperparameter tuning. In comparison to the algorithms' default values, both the TimeEval approach and HYPEX's parameter

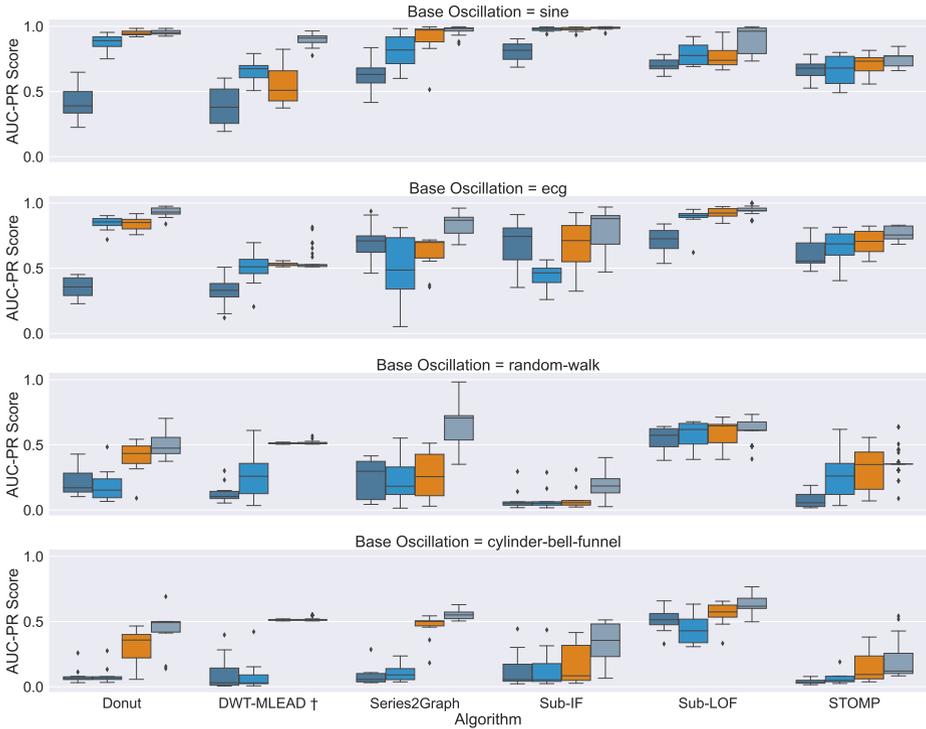


Fig. 5: Distribution of maximum AUC-PR scores achieved by ■ *Default Parameters*, ■ *Timeeval Heuristics*, ■ *Parameter Model*, and ■ *Full Optimization* per base oscillation and algorithm. † Empty parameter model for base oscillations *random walk* and *cylinder-bell-funnel*.

model achieve high AUC-PR scores; the difference is that no manual work was needed to find the parameter model. Furthermore, HYPEX’s performance scores get even close to the maximum scores achieved via Bayesian optimization (consider the highest point for comparison); as an unsupervised parametrization approach, though, HYPEX does not require labels for the input time series to optimize it, but instead generates training data automatically.

We summarize our evaluation results across all six tested algorithms on four base oscillations in Fig. 5. A single box plot shows the distribution of 10 AUC-PR scores. Each score represents the evaluation result on a single evaluation time series dataset. To represent the Bayesian full optimization trials, we pick the maximum AUC-PR scores per time series dataset obtained in each of the 300 trial runs. As expected and confirmed in this experiment, a full optimization for a target dataset with training data performs best on all dataset types, i.e., base oscillations. Again, the algorithms’ default parameters achieve the worst performance

scores in the majority of the conducted experiments. The manual hyperparameter settings and heuristics of TimeEval mostly deliver higher performance scores than the algorithms’ default parameters. In some cases, such as Donut on random walk, Series2Graph on ECG and random walk, and Sub-IF on ECG, however, the TimeEval efforts could not predict better-performing hyperparameters than the default parameters. With no human effort and no pre-labeled training data, our parameter models’ hyperparameter suggestions surpass both default parameterization and TimeEval performances in most experiments. Even in cases where HYPEX did not discover any parameter rules for an anomaly detector and simply suggested fixed values, these values still outperformed the detector’s default parameters (see DWT-MLEAD on base oscillations sine, random walk, and cylinder bell funnel).

In terms of absolute performance, we see that all evaluated anomaly detectors tend to perform best on cyclical base oscillations, such as sine and ECG, and struggle with non-cyclical ones, such as random walk and cylinder bell funnel.

### 4.3 Sensitivity to Thresholds

HYPEX uses two thresholds, namely  $\alpha$  and  $\beta$ , to adjust the minimum confidence scores of the (in-)dependence and CI tests. In Sect. 3.3.5, we discussed how the algorithm automatically determines the optimal values for  $\alpha$  and  $\beta$  during the parameter model selection. Fig. 6 shows the parameter model results on the evaluation datasets when HYPEX performs the optimal threshold determination on the validation datasets. We optimized each parameter model’s fixed parameters independently for each of the threshold tuples. Many experiments indicate that the full optimization trial runs have only low variances in optimal parameter values across different evaluation datasets. Thus, parameter rules do not showcase their full potential, as optimized fixed parameters achieve similar high-performance scores. However, the full optimization’s best parameter value suggestions for the algorithm STOMP on the base oscillation sine showed high variances across the different datasets. The  $(\alpha, \beta)$  threshold tuple  $(0.46, 0.2)$  resulted in an empty causal graph, thus the suggested parameters on the evaluation datasets are based on fixed values only. However, it achieves the second-best detection results on average among the compared parameter models. We also see that using fixed values only comes at the cost of increased variance across the evaluation datasets. Hence, HYPEX automatically chooses

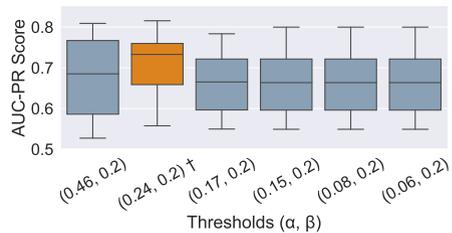


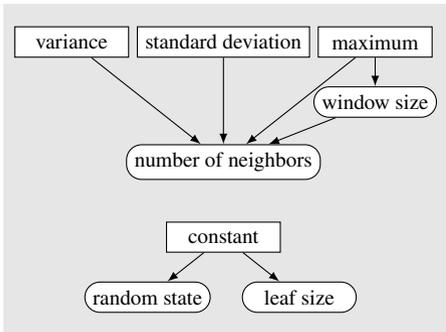
Fig. 6: Comparison of AUC-PR score distributions by selected  $\alpha$  and  $\beta$  threshold values on 10 evaluation datasets for algorithm STOMP [Zh16] and base oscillation sine. Each named threshold tuple represents a set of tuples that all result in the identification of identical parameter-rules.

† The threshold value HYPEX automatically selected (see Sect. 3.3.5).

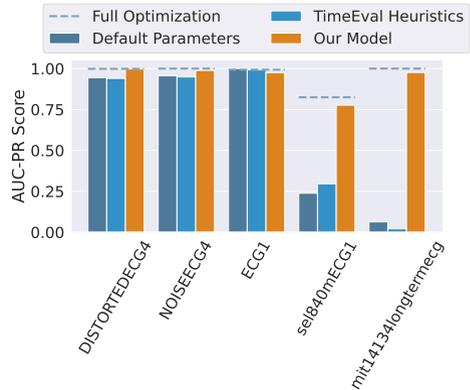
the threshold values (0.24, 0.2). Across all experiments, the resulting parameter models contain a mixture of parameter rules and fixed hyperparameter values, which is a result that performs best on the evaluation datasets and shows to have a lower variance than using only fixed parameter values.

### 4.4 Application on real-world data

In this section, we evaluate how HYPEX’ hyperparameter suggestions perform on five real-world datasets taken from the KDD-TSAD benchmark collection [Ke21]. All five time series represent ECG signals with varying properties, and contain different anomalies. Due to the space limitation and because Sub-LOF performed consistently well in all our previous evaluations, we restrict our experiment to the Sub-LOF algorithm. We configured HYPEX to optimize all four hyperparameters of Sub-LOF, which are window size, number of neighbors, leaf size, and random state, and to create datasets with similar data characteristics to the target datasets, i. e., ECG shaped data. We allow HYPEX to mutate base oscillation frequency, noise level, amplitude, mean, and anomaly details, such as position, length, and anomaly shift size. For both the generated training datasets and the real-world datasets, HYPEX uses *tsfresh* [Ch18] to extract 12 time series features as the dataset characteristics.



□ dataset characteristic    ○ hyperparameter



(a) HYPEX parameter model learned for the Sub-LOF algorithm on the synthetically generated training data. (b) AUC-PR score of Sub-LOF on real datasets. HYPEX was trained only on synthetic data.

Fig. 7: HYPEX’ parameter model and AUC-PR scores for Sub-LOF on real ECG datasets.

Fig. 7a shows HYPEX’ final parameter model for Sub-LOF. The optimization on the synthetic training datasets identified dependencies for the hyperparameters window size and number of neighbors: While window size depends solely on the dataset characteristic maximum, number of neighbors depends not only on dataset characteristics, but also on the hyperparameter window size. HYPEX assigned constant values to the parameters

random state and leaf size because they had no significant influence on the algorithm’s performance on the training datasets.

In Fig. 7b, we show the AUC-PR scores of Sub-LOF on the five real-world datasets using (a) HYPEX’ parameter model, (b) the algorithms’ default parameters, (c) the parameter recommendations of TimeEval, and (d) the maximum of a full optimization run with 300 trials. The full optimization run indicates the optimal performance that Sub-LOF could achieve on each dataset when taking the ground truth into account. Both the default hyperparameters and the TimeEval hyperparameters perform poorly for the datasets *sel840mECG1* and *mit14134longtermecg*. For the dataset *ECG1*, the AUC-PR scores using the default values (1.00) and the hyperparameter values from TimeEval (1.00) are marginally higher than HYPEX’ score (0.98). HYPEX’ hyperparameter values, however, can achieve an AUC-PR score close to the full optimization run for all datasets. This demonstrates HYPEX’s capability to learn a parameter model on synthetic training datasets that can significantly outperform alternative parametrization strategies, and that routinely approaches the maximum achievable score.

## 5 Conclusion

In this paper, we addressed the time-consuming process of tuning hyperparameters. Our proposed system HYPEX extracts parameter rules that can be used to transfer knowledge about the relationship (a) between parameters and data characteristics, and (b) between two parameters to yet unseen application data. While previous work included manually crafted heuristics [SWP22] or long-running optimization tasks [DC21; Le12; PGC+99; Sh15], which both require labels on large test datasets, our work proposes an automated approach using synthetic datasets to derive parameter calculation rules based on identified causal relationships. In our evaluation, HYPEX’s parameter suggestions outperformed the anomaly detectors’ default parameters as well as hand-crafted heuristics across different anomaly detection methods and base oscillations. We showed that identified fixed parameters perform well on a variety of different datasets at the cost of higher variance. HYPEX’s approach, using a mixture of parameter rules and fixed hyperparameter values with the automatic parameter model selection, predicts well-performing, reliable hyperparameter values on different datasets. Future work includes the extension of HYPEX to categorical data types and its application and evaluation in other domains, such as data cleaning or pattern mining.

### Acknowledgements

This paper is based on the excellent work of Mats Pörschke, whose career prevented him from officially co-authoring this publication. His master thesis<sup>6</sup> explains HYPEX in more detail. We thank Mats for his effort and permitting the presentation of the results.

---

<sup>6</sup> [https://github.com/HPI-Information-Systems/hypex/raw/main/masterthesis\\_hypex.pdf](https://github.com/HPI-Information-Systems/hypex/raw/main/masterthesis_hypex.pdf)

## References

- [Ak19] Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD). 2019.
- [BB12] Bergstra, J.; Bengio, Y.: Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)* 13/1, 2012.
- [Be11] Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B.: Algorithms for hyper-parameter optimization. In: *Advances in Neural Information Processing Systems (NIPS)*. 2011.
- [BP20] Boniol, P.; Palpanas, T.: Series2Graph: Graph-Based Subsequence Anomaly Detection for Time Series. *Proceedings of the VLDB Endowment* 13/12, 2020.
- [Br00] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J.: LOF: identifying density-based local outliers. In: *Proceedings of the International Conference on Management of Data (SIGMOD)*. 2000.
- [Br97] Bradley, A. P.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition* 30/7, 1997.
- [Ch18] Christ, M.; Braun, N.; Neuffer, J.; A.W., K.-L.: Time Series Feature Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing* 307/1, 2018.
- [DC21] Dahmen, J.; Cook, D. J.: Indirectly Supervised Anomaly Detection of Clinically Meaningful Health Events from Smart Home Data. *ACM Transactions on Intelligent Systems and Technology (TIST)* 12/2, 2021.
- [DG06] Davis, J.; Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2006.
- [DMC16] Dewancker, I.; McCourt, M.; Clark, S.: *Bayesian Optimization for Machine Learning: A Practical Guidebook*, 2016, arXiv: [cs/1612.04858](https://arxiv.org/abs/1612.04858).
- [FH19] Feurer, M.; Hutter, F.: *Hyperparameter Optimization*. In: *Automatic Machine Learning: Methods, Systems, Challenges*. Springer Berlin Heidelberg, 2019.
- [Fr18] Frazier, P. I.: *A Tutorial on Bayesian Optimization*, 2018, arXiv: [stat.ML/1807.02811](https://arxiv.org/abs/1807.02811).
- [GZS19] Glymour, C.; Zhang, K.; Spirtes, P.: Review of causal discovery methods based on graphical models. *Frontiers in Genetics* 10/1, 2019.
- [HHL14] Hutter, F.; Hoos, H.; Leyton-Brown, K.: An Efficient Approach for Assessing Hyperparameter Importance. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2014.

- [Hi12] Hinton, G. E.: A practical guide to training restricted Boltzmann machines. In: *Neural networks: Tricks of the trade*. Springer Berlin Heidelberg, 2012.
- [Ke21] Keogh, E.; Dutta Roy, T.; Naik, U.; Agrawal, A.: Multi-dataset Time-Series Anomaly Detection Competition, 2021, URL: <https://compete.hexagonml.com/practice/competition/39/>, visited on: 11/09/2021.
- [Le12] LeCun, Y. A.; Bottou, L.; Orr, G. B.; Müller, K.-R.: Efficient backprop. In: *Neural networks: Tricks of the trade*. Springer Berlin Heidelberg, 2012.
- [Li15] Liu, D.; Zhao, Y.; Xu, H.; Sun, Y.; Pei, D.; Luo, J.; Jing, X.; Feng, M.: Opprentice: Towards practical and automatic anomaly detection through machine learning. In: *Proceedings of the Internet Measurement Conference (IMC)*. 2015.
- [LTZ08] Liu, F. T.; Ting, K. M.; Zhou, Z.-H.: Isolation forest. In: *IEEE International Conference on Data Mining (ICDM)*. 2008.
- [Pe14] Peters, J.; Mooij, J. M.; Janzing, D.; Schölkopf, B.: Causal discovery with continuous additive noise models. *Journal of Machine Learning Research (JMLR)* 15/1, 2014.
- [PGC+99] Pelikan, M.; Goldberg, D. E.; Cantú-Paz, E., et al.: BOA: The Bayesian optimization algorithm. In: *Proceedings the Genetic and Evolutionary Computation Conference (GECCO)*. 1999.
- [Ro15] Rocklin, M.: Dask: Parallel computation with blocked algorithms and task scheduling. In: *Proceedings of the Python in Science Conference (SciPy)*. 2015.
- [Sh15] Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; De Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. In: *Proceedings of the IEEE*. 2015.
- [Sp00] Spirtes, P.; Glymour, C. N.; Scheines, R.; Heckerman, D.: *Causation, prediction, and search*. MIT press, 2000.
- [Su17] Sun, J.; Ajwani, D.; Nicholson, P. K.; Sala, A.; Parthasarathy, S.: Breaking Cycles In Noisy Hierarchies. In: *Proceedings of the ACM Web Science Conference (WebSci)*. 2017.
- [SWP22] Schmidl, S.; Wenig, P.; Papenbrock, T.: Anomaly Detection in Time Series: A Comprehensive Evaluation. *Proceedings of the VLDB Endowment* 15/9, 2022.
- [TKB17] Thill, M.; Konen, W.; Bäck, T.: Time series anomaly detection with discrete wavelet transforms and maximum likelihood estimation. In: *International Conference on Time Series (ITISE)*. 2017.
- [WSP22] Wenig, P.; Schmidl, S.; Papenbrock, T.: TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms. *Proceedings of the VLDB Endowment* 15/12, 2022.

- [Xu18] Xu, H.; Chen, W.; Zhao, N.; Li, Z.; Bu, J.; Li, Z.; Liu, Y.; Zhao, Y.; Pei, D.; Feng, Y., et al.: Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In: Proceedings of the International World Wide Web Conference (WWW). 2018.
- [ZG07] Zesch, T.; Gurevych, I.: Analysis of the Wikipedia category graph for NLP applications. In: Proceedings of the Second Workshop on TextGraphs: Graph-Based Algorithms for Natural Language Processing. 2007.
- [Zh16] Zhu, Y.; Zimmerman, Z.; Senobari, N. S.; Yeh, C.-C. M.; Funning, G.; Mueen, A.; Brisk, P.; Keogh, E.: Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. In: IEEE International Conference on Data Mining (ICDM). 2016.
- [Zh17] Zhang, Q.; Filippi, S.; Flaxman, S.; Sejdinovic, D.: Feature-to-feature regression for a two-step conditional independence test. In: Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI). 2017.



# Evolution of Degree Metrics in Large Temporal Graphs

Christopher Rost,<sup>1</sup> Kevin Gomez,<sup>1</sup> Peter Christen,<sup>2</sup> Erhard Rahm<sup>1</sup>

**Abstract:** Graph metrics, such as the simple but popular vertex degree and others based on it, are well defined for static graphs. However, adapting static metrics for temporal graphs is still part of current research. In this paper, we propose a set of temporal extensions of four degree-dependent metrics, as well as aggregations like minimum, maximum, and average degree of (i) a vertex over a time interval and (ii) a graph at a specific point in time. We show why using the static degree can lead to wrong assumptions about the relevance of a vertex in a temporal graph and highlight the need to include *time* as a dimension in the metric. We propose a baseline algorithm to calculate the degree evolution of all vertices in a temporal graph and show its implementation in a distributed in-memory dataflow system. Using real-world and synthetic datasets containing up to 462 million vertices and 1.7 billion edges, we show the scalability of our algorithm on a distributed cluster achieving a speedup of around 12 on 16 machines.

**Keywords:** Temporal Property Graph; Temporal Degree; Degree Evolution; Temporal Graph Metric

## 1 Introduction

Temporal graphs are graphs that change in structure and content over time, where changes are captured and maintained as part of the graph data model. Many approaches exist to formally define a temporal graph [Iy21, Ko09, Ro22, HR21]. A graph's evolution is either represented as a series of snapshots, or by vertex and edge annotations for timestamps or time intervals describing their validity. These extended graph models allow analyzing the current or a past state of a graph as well as the evolution of the graph. Examples for temporal graph analysis are the exploration of human contact networks to detect the transmission of a disease [SK05, RKC01] or analyzing the change in the utilization of bike rental stations [Li15, Tl20]. In such graphs, the concepts of graph metrics also change because time is added as a new dimension. Metrics used for the characterization of static graphs need to be redefined or extended to take temporal evolution into account [Ni13].

One simple yet important metric of a vertex is the *vertex degree* [GY03]. It is determined by the number of incoming and outgoing edges (which is, except for multigraphs, equal to the number of neighbors) and thus a simple indicator for the relevance or importance of a vertex in a static graph. A vertex with a high degree can be seen as a strongly connected vertex, whereas a vertex with a degree of zero is an isolated vertex or singleton. The vertex degree is also known as the centrality measure *degree centrality* [Fr78], that can be used to find,

---

<sup>1</sup> University of Leipzig & ScaDS.AI Dresden/Leipzig, Germany. {rost,gomez,rahm}@informatik.uni-leipzig.de

<sup>2</sup> Australian National University, Canberra, Australia. peter.christen@anu.edu.au

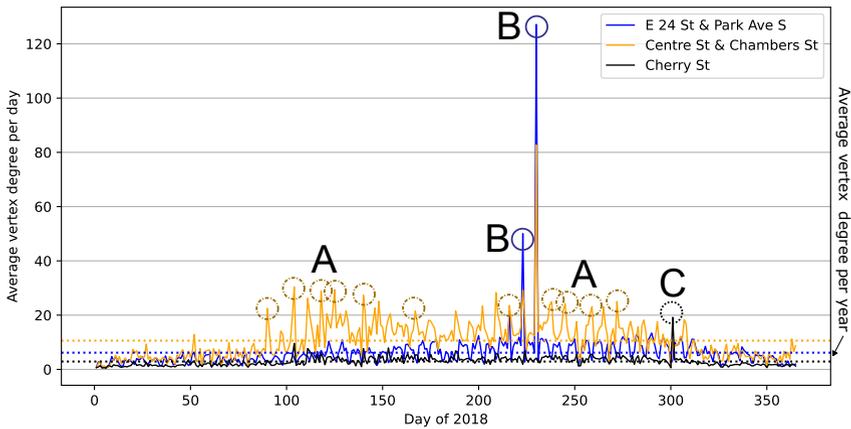


Fig. 1: Degree evolution of selected rental stations in NYC for 2018. For each day, the average degree is plotted. **A** indicates peaks on weekends, **B** a construction embargo event and **C** a Halloween parade.

for example, popular people according to their number of friendships in a social network, or the stations with the highest throughput of bike rentals in a bike-sharing network.

The minimum and maximum degrees are metrics that describe the vertices with the smallest and largest numbers of connections, respectively. The *degree range* [LJ21], *degree variance* [LJ21, Sn81, SE20] and the *average nearest neighbor degree* (ANND) [LJ21, YvdHL17], are aggregate metrics that can reveal important graph and vertex characteristics. The *degree range* of a graph (the difference between the maximum and minimum degree) describes the connectivity gap between the best and least connected vertices. For a bike-sharing network, a small degree range indicates a good distribution of rental stations without any hardly visited stations, whereas a high degree range indicates irregular usage. Another extended measure of a graph's heterogeneity is the *degree variance*, where a high variance shows a high inequality in the connectivity of the vertices. The *ANND*, on the other hand, reveals if a vertex is connected to others with a high connectivity, e.g., a social network user who is mainly friend with other users who are strongly connected.

Using only the static vertex degree is of limited value in an evolving graph as it cannot reflect the impact of topology changes. The same restriction applies for static aggregated metrics such as the average degree value [KA12] or the sum of all degrees [TBF17]. There is no information about *when* a vertex has what degree, *how long* this degree is valid, and *when* it increases or decreases. This is important, for example, in a bike-sharing network where vertices represent stations and directed edges connect the start and return stations of bike rentals.

Fig. 1 shows the time series representing the evolution of the vertex degree of three selected bike rental stations in NYC for 2018, calculated from the publicly available dataset also used

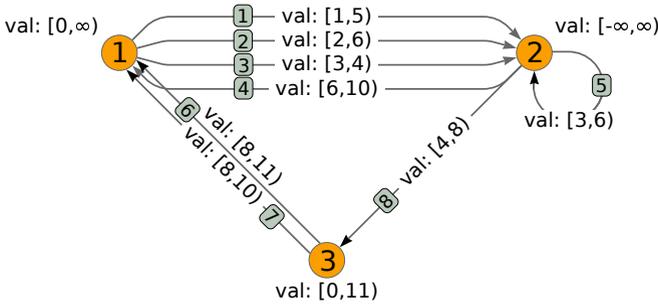


Fig. 2: An example temporal graph.

in our evaluations (see Sect. 6). For example, one can see the popularity of the station at Centre St & Chambers St on weekends by periodic peaks (marked with **A**) or the significantly higher rental rate of two stations during the Summer Streets Construction Embargo<sup>3</sup> in August (marked with **B**). Further, the impact of a Halloween parade<sup>4</sup> on Cherry St. (marked by **C**) is visible in these time-series. This shows that there are stations that are generally popular, such as in a city center or near train stations, as well as stations that are only popular at certain times, e.g., on weekends or during events. Further, comparing stations using the static or aggregated metrics, which are shown in Fig. 1 as dotted lines, may lead to the assumption that they seem equal by sharing a similar degree value, which in fact is not true over time which can be revealed by temporal metrics.

Fig. 2 shows a toy example of a temporal graph, which we use to illustrate the problem further. Each vertex and directed edge has a unique numeric identifier and a left-close right-open time interval  $[\omega_a, \omega_b)$ <sup>5</sup> assigned. For example, the edge with identifier 5 (hereinafter referred to as  $e_5$ ) is valid from time point 3 (in the following denoted as  $\omega_3$ ) to  $\omega_6$ , whereas the vertex  $v_1$  is valid from  $\omega_0$  to the maximum upper bound, denoted by the infinity symbol  $\infty$  ( $\omega_{max}$ ).

From a static perspective, if we disregard the graph’s evolution, we can see that the vertex degrees are  $deg(v_1) = 6$ ,  $deg(v_2) = 7$ , and  $deg(v_3) = 3$ . However, if time is considered, then the degree values change continuously so that the evolution of the degree value forms a time series. For example, at time  $\omega_1$ , the degree of  $v_1$  is 1, and the same at time  $\omega_5$ . Further, since  $v_1$  is valid until forever and the last validity of its edges end at time  $\omega_{11}$  (exclusive), the degree from  $\omega_{11}$  to forever ( $\omega_{max}$ ) is 0. Fig. 3 exemplifies the evolution of the degrees of  $v_1, v_2$  and  $v_3$ , inclusive in- and outdegree of  $v_1$  ( $deg^-(v_1)$  and  $deg^+(v_1)$ ).

It can be seen that the maximum degree of vertex  $v_1$  is only 3 over its entire period of validity. From the vertex lower bound  $\omega_0$  to time  $\omega_1$ , the degree is 0 – the same from  $\omega_{11}$

<sup>3</sup><https://www.milrose.com/insights/2018-summer-streets-construction-embargo> (visited 2022-11-01)

<sup>4</sup><https://patch.com/new-york/east-village/halloween-dog-parade-2018-what-you-need-know> (visited 2022-11-01)

<sup>5</sup>For simplicity we use integer interval bounds. It holds  $[\omega_a, \omega_b) := \{\omega \in \mathbb{N} : \omega_a \leq \omega < \omega_b\}$ .

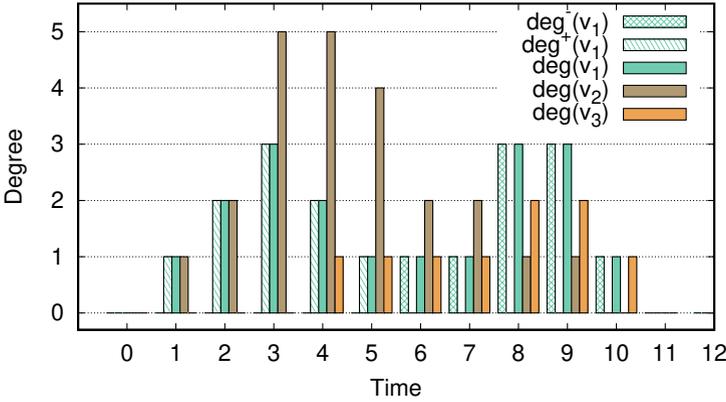


Fig. 3: Degree evolution of vertex  $v_1, v_2$  and  $v_3$  from  $\omega_0$  to  $\omega_{12}$ . In addition, the indegree  $deg^-(v_1)$  and outdegree  $deg^+(v_1)$  are given for  $v_1$ .

onwards. Compared to the static point of view, where the degree is 6 for  $v_1$ , we can see that during the evolution of the graph the vertex never reaches this value. The same holds for the bike rental example of Fig. 1. For example, the static degree of the station “Cherry St.” is 50, whereas the maximum value over the year is just 20 for a single day. This shows the importance of considering the changes of the degree metric over time. The use of the static degree metric for assessing the importance of a vertex can lead to misinterpretations, whereas using the degree evolution provides the exact degree for any time in the lifetime of the graph.

**Contributions:** In this work we focus on four time-sensitive degree-dependent graph measures: the vertex degree itself and its aggregations, the degree range, the degree variance, and the average nearest neighbor degree. We extend these well known static metrics with a time dimension and establish two new formal definitions per metric: (i) a temporal version which defines the metric at a specific point in time, and (ii) an evolutionary version which defines the change of the metric within a time interval as a time series. We then present a baseline algorithm that can calculate the degree evolution for all vertices of a given temporal graph. Using a binary search tree called *degree tree*, the algorithm efficiently maintains the degree changes of each vertex. We show how our algorithm can be adapted to a distributed processing model, which is further illustrated by the implementation as a graph analysis operator using a distributed in-memory dataflow system. In our experiments, we evaluate the scalability of our implementation which shows a sublinear growth of runtime by increasing dataset size as well as a speedup of up to 12 on a cluster with 16 physical machines.

## 2 Related work

Some works have defined a degree metric for vertices in a temporal graph, mainly by expanding the static version for temporal graphs. Thompson et al. [TBF17] introduce a *temporal degree centrality* metric for the domain of network neuroscience. They show that a node's influence in a temporal network can be represented by the centrality metric, which is the sum of the number of edges across a series of time points. If an edge is valid for multiple time points, it will be counted multiple times. However, this approach does not quantify the temporal order of edges so that different vertices with identical metrics cannot be distinguished.

A similar definition of temporal degree centrality is given by Long et al. [Lo20] and Wu et al. [Wu14]. Both calculate the sum of degrees over a time interval, which provides an estimate of a node's centrality in a temporal network. Wang et al. [Wa17] propose the *temporal degree deviation centrality* metric that can be calculated from a temporal network using graph snapshots. A similar approach defines the temporal degree as the number of nodes to which a vertex is linked in all timestamps of an interval without interruption [Ci20].

The *time-ordered graph* model by Kim et al. [KA12] can represent a dynamic network with a fixed vertex set and interval edges. For graphs of this model, several centrality metrics were introduced (including degree) to include the graph's temporal characteristic. Temporal degree is defined as the degree  $D_{i,j}(v)$  for a vertex  $v \in V$  in a time interval  $[i, j]$ . Tebaldinova et al. [TI20] use the degree as a temporal measure of centrality for bike-sharing stations. They show that the changing degree determines the time-distributed intensity of incoming and outgoing bike flows at a station.

In all these related works, the temporal degree is mostly seen as a scalar, aggregated (summed) value over a certain time interval, that is used as a centrality measure. In our approach, described next, we define both a *temporal degree* at a specific point in time as well as *degree evolution* for a time interval as a time series. This allows exact statements when a metric has what value for how long. In addition, our data model allows both changes in vertices and edges, as we describe in Sect. 3.1.

## 3 Degree-dependent metric evolution

We first define the temporal graph data model we use as a basis for our work, and then introduce new temporal notations of degree-dependent metrics for vertices in Sect. 3.2, and metrics for a whole temporal graph in Sect. 3.3.

### 3.1 Temporal graph model

We use a simplified version of the *Temporal Property Graph Model (TPGM)* data model [Ro22]<sup>6</sup>. Although the model supports bitemporal versioning, for simplicity we limit ourselves to one time dimension. Thus, vertices and edges are assigned with a left-closed right-open time interval to represent the element's validity according to application-specific valid-time. Unlike most temporal graph models [Ca21], not only the edge set is dynamic, but the vertex set can also change over time. Contact sequence graphs [Ho18] can also be modeled by representing the time  $\omega_i$  of the contact as time interval  $[\omega_i, \infty)$ ,  $[\omega_i, \omega_{i+1})$  or  $[\omega_i, \omega_j)$  (depending on the use-case), where  $\omega_j$  is the time of a subsequent contact. A TPGM graph is formally defined as follows:

**Definition 1 (TPGM graph [Ro22, GS20])** *A TPGM graph is a directed multigraph  $\mathcal{G} = (V, E, \Omega)$  with the following specifications:*

*$V$  is a finite set of vertices. Each vertex  $v \in V$  is a tuple  $\langle v_{id}, \tau \rangle$ , where  $v_{id}$  is a unique vertex identifier,  $\tau$  is a time-interval of the form  $[\omega_{start}, \omega_{end})$  for which the vertex is valid with respect to  $\Omega$  (defined below). We constrain that each  $v \in V$  has at least one edge throughout the graph history, i.e., vertices that were isolated over the entire graph lifetime are not part of  $V$ .*

*$E$  is a finite set of edges. Each edge  $e \in E$  is a tuple  $\langle e_{id}, s_{id}, t_{id}, \tau \rangle$ , where  $e_{id}$  is a unique edge identifier that allows multiple edges between the same nodes,  $s_{id}$  and  $t_{id}$  are the source and target vertex identifier,  $\tau$  is the time-interval of the form  $[\omega_{start}, \omega_{end})$  for which the edge is valid with respect to  $\Omega$ .*

*$\Omega$  represents the valid-time domain where an instant in time is a time point  $\omega_i$  with limited precision, e.g., milliseconds. A time interval  $\tau = [\omega_{start}, \omega_{end})$  with  $\omega_{start}, \omega_{end} \in \Omega$  starts at  $\omega_{start}$  and ends at  $\omega_{end}$ . Since it is a left-close right-open interval, it includes  $\omega_{start}$  but excludes  $\omega_{end}$ .*

We refer to our previous work [Ro22], in which several constraints are defined to ensure a consistent TPGM graph. Since the set of nodes  $V$  and edges  $E$  changes over time, we introduce two time-dependent sets of nodes and edges that we use later in the formal definitions in Sect. 3.2 and Sect. 3.3:

- $V(\omega_i) \subseteq V$  is a finite subset of vertices, where each vertex is valid at the given time point  $\omega_i$ , i. e., for all  $v = \langle v_{id}, \tau \rangle \in V(\omega_i)$  with  $\tau = [\omega_{start}, \omega_{end})$  it holds:  $\omega_{start} \leq \omega_i < \omega_{end}$ .

<sup>6</sup>A TPGM graph, in addition, formally defines the concept of so-called logical graphs and assigns type labels and properties (key-value pairs) to nodes, edges, and logical graphs. Since neither is relevant for this work, we excluded it for the sake of simplicity.

- $E(\omega_i) \subseteq E$  is a finite subset of edges, where each edge is valid at the given time point  $\omega_i$ , i. e., for all  $e = \langle e_{id}, s_{id}, t_{id}, \tau \rangle \in E(\omega_i)$  with  $\tau = [\omega_{start}, \omega_{end})$  it holds:  $\omega_{start} \leq \omega_i < \omega_{end}$ .
- $G(\omega) = (V(\omega), E(\omega))$  is a graph snapshot (or state) of a temporal graph  $G$  at a specific point in time  $\omega$ .

### 3.2 Vertex-centric temporal degree metrics

For each of the following degree-based metrics, we first refer to the static version and then introduce our temporal and evolutionary version of the respective metric.

**Vertex degree and aggregations.** According to graph theory [GY03, Di10], the static (non-temporal) **vertex degree**  $deg(v)$  is formally defined as follows:

**Definition 2 (Vertex degree [GY03, Di10])** *The **degree** (or valence) of a vertex  $v$  in a static graph  $G = (V, E)$ , denoted  $deg(v)$ , is the number of proper edges incident to  $v$  plus twice the number of self-loops. Simplified, the degree of a vertex is the number of its edges. The **indegree** of a vertex  $v$ , denoted as  $deg^-(v)$ , is the number of edges directed to  $v$  whereas the **outdegree** of vertex  $v$ , denoted as  $deg^+(v)$ , is the number of edges directed from  $v$ . Each self-loop at  $v$  counts one toward the indegree of  $v$  and one toward the outdegree.*

Having a static view on the graph of Fig. 2, example vertex degrees are  $deg(v_1) = 6$ ,  $deg(v_2) = 7$ ,  $deg^+(v_2) = 3$ , and  $deg^-(v_3) = 1$ .

For temporal graphs, we now define the *temporal degree* as the degree of a vertex *at a specific point in time*.

**Definition 3 (Temporal degree)** *The **temporal degree** of a vertex  $v$  in a temporal graph  $G = (V, E)$ , denoted as  $degt(v, \omega)$ , is the degree of that vertex at time  $\omega$  in the graph snapshot  $G(\omega)$ . It is defined as:*

$$degt(v, \omega) \begin{cases} deg(v), & \text{if } v \in V(\omega), \\ \text{not defined,} & \text{otherwise.} \end{cases} \quad (1)$$

*If  $v \notin V(\omega)$ , the degree is not defined. Analogous to the static degree, the **temporal indegree**  $degt^-(v, \omega)$  is the number of edges directed to  $v$ , and **temporal outdegree**  $degt^+(v, \omega)$  is the number of edges directed from  $v$ , at time  $\omega$ .*

For example, in the graph of Fig. 2, the temporal degree of vertex  $v_1$  at time  $\omega_4$  is  $degt(v_1, \omega_4) = 2$ , whereas the temporal indegree of vertex  $v_1$  at time  $\omega_8$  is  $degt^-(v_1, \omega_8) = 3$ . There are clear differences between the static compared to the temporal metrics.

From the perspective of a vertex  $v$ , the degree of that vertex changes according to the existence of neighbours of  $v$ . For a given time interval  $\tau$ , we thus define the *degree evolution* as a time series of temporal degrees, which contains all degree values with their corresponding time in the given interval.

**Definition 4 (Degree evolution)** *The **degree evolution**  $degev(v, \tau) := \{x_1, x_2, \dots, x_m\}$  of a vertex  $v$  is a time series of elements  $x_i := deg_t(v, \omega)$ , with  $1 \leq i \leq m$  and  $m = \omega_{end} - \omega_{start}$ . Each  $x_i$  represents a temporal degree at time  $\omega_j$ , i.e.,  $x_1$  at time point  $\omega_{start}$  and  $x_m$  at  $\omega_{end} - 1$ , for the interval  $\tau = [\omega_{start}, \omega_{end})$ . Further, the temporal degree is a special case of the degree evolution:  $degev(v, \tau) = \{deg_t(v, \omega_i)\}$  with  $\tau = [\omega_i, \omega_{i+1})$  as an interval with a single time point. Furthermore,  $degev^+(v, \tau)$  denotes the **outdegree evolution** whereas  $degev^-(v, \tau)$  denotes the **indegree evolution**.*

For our example graph of Fig. 2, the degree evolution of vertex  $v_1$  in the interval  $\tau = [\omega_0, \omega_{11})$  is  $degev(v_1, \tau) = \{0, 1, 2, 3, 2, 1, 1, 1, 3, 3, 1\}$ .

The degree evolution defines the development of a vertex degree over a given time interval. This can now be used to determine the minimum, maximum and average degree of a *vertex* over a time interval, i.e., a vertex-centric aggregation.

**Definition 5 (Vertex-centric min/max/avg degree)** *The **vertex-centric minimum degree** of a vertex  $v$  within a time interval  $\tau$  is the smallest value of all temporal degrees of  $v$  in this interval. Similarly, the **vertex-centric maximum degree** is the largest value and the **vertex-centric average degree** is the average value over all time points  $\omega \in \tau$ . With  $|\tau|$  as the number of all time points in the interval  $\tau$  holds:*

$$deg_{min}(v, \tau) := \min\{deg_t(v, \omega) | \forall \omega \in \tau\}, \quad (2)$$

$$deg_{max}(v, \tau) := \max\{deg_t(v, \omega) | \forall \omega \in \tau\}, \quad (3)$$

$$deg_{avg}(v, \tau) := \frac{1}{|\tau|} \sum_{\omega \in \tau} deg_t(v, \omega). \quad (4)$$

**Average Nearest Neighbor Degree.** An analyst may be interested in whether entities in a graph tend to connect to others with a high connectivity, or, the opposite case, connections occur randomly and irrespective of the degree [LJ21]. The former situation is referred to as *preferential attachment* in network science [JNB03] and applies to many real-world networks [Ne01, Ca06], including evolving networks [JNB03]. A metric to measure this tendency is the *average nearest neighbor degree* (ANND)  $deg_{nn}(v)$ . For a vertex  $v$ , the ANND is the sum of the direct neighbor degrees divided by the degree of  $v$ .

**Definition 6 (Average nearest neighbor degree [LJ21])** *The average nearest neighbor degree  $deg_{nn}(v_i)$  of a vertex  $v_i$  of a static graph  $G$  is defined as the sum of the degrees of each of the vertex' neighbor  $v_j$  divided by the degree of  $v_i$ :*

$$deg_{nn}(v_i) := \frac{1}{deg(v_i)} \sum_{v_j \in N(v_i)} deg(v_j). \tag{5}$$

The set  $N(v_i) \subset V$  is defined as the set of vertices incident to a vertex  $v_i$  (its neighbors).

From a static perspective of the example graph of Fig. 2, the ANNDs are  $deg_{nn}(v_1) = \frac{deg(v_2)+deg(v_3)}{deg(v_1)} = 1.67$ ,  $deg_{nn}(v_2) = 1.43$  and  $deg_{nn}(v_3) = 4.34$ . These results suggest that vertex  $v_3$  seems to have the strongest tendencies to connect to others who are also popular, while  $v_1$  and  $v_2$  display weaker tendencies. The average degree of the graph (here  $deg_{avg} = 5.34$ ) can be used to interpret an ANND value. The larger the value compared to the average degree of the graph, the more likely we can assume that its neighbors are more popular than average. As the other degree-dependent metrics, the ANND will change over time if a graph evolves. To calculate the ANND of a vertex at a specific point in time, we now define the *temporal average nearest neighbour degree*:

**Definition 7 (Temporal ANND)** *The temporal average nearest neighbor degree (TANND)  $degt_{nn}(v_i, \omega)$  of a vertex  $v_i$  is defined as the sum of the temporal degrees of each of the vertex' neighbor (at time  $\omega$ ) divided by the temporal degree of  $v_i$ . Furthermore, the set  $N(v_i, \omega) \subset V(\omega)$  is defined as the set of neighbors of vertex  $v_i$  at time  $\omega$ . It then holds:*

$$degt_{nn}(v_i, \omega) := \frac{1}{degt(v_i, \omega)} \sum_{v_j \in N(v_i, \omega)} degt(v_j, \omega). \tag{6}$$

For the example in Fig. 2, the TANND for  $v_1$  at time  $\omega_4$  is  $degt_{nn}(v_1, \omega_4) = 2.5$ , while at time  $\omega_9$  it is  $degt_{nn}(v_1, \omega_9) = 1$ .

An analyst may be also interested in the evolution of the ANND over a time interval, like how people's propensity to rent a bike from one popular location and ride to another popular location is changing within a month. We introduce the *average nearest neighbor degree evolution* to define a series of TANND values within a time interval.

**Definition 8 (ANND evolution)** *The average nearest neighbor degree evolution (ANNDE)  $dege_{nn}(v, \tau) := \{x_1, x_2, \dots, x_m\}$  of a vertex  $v$  is a time series of elements  $x_i := degt_{nn}(v, \omega)$ , with  $1 \leq i \leq m$  and  $m = \omega_{end} - \omega_{start}$ . Each  $x_i$  represents the TANND at time  $\omega_j$ , i.e.,  $x_1$  at time point  $\omega_{start}$  and  $x_m$  at  $\omega_{end} - 1$ , for the interval  $\tau = [\omega_{start}, \omega_{end})$ . The TANND is a special case of the ANNDE:  $dege_{nn}(v, \tau) = \{degt_{nn}(v, \omega)\}$ , with  $\tau = [\omega_i, \omega_{i+1})$  as an interval with a single time point.*

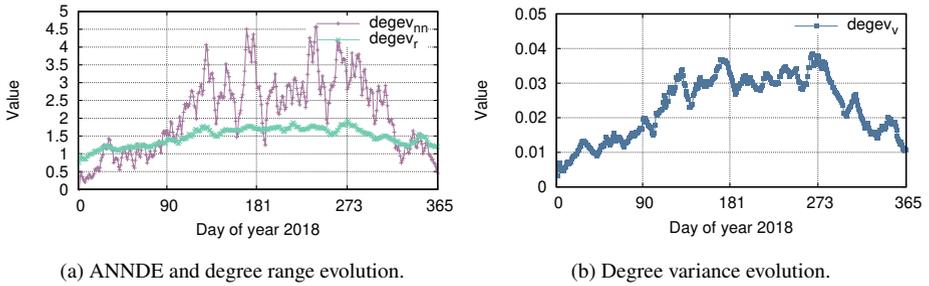


Fig. 4: Resulting time-series of selected degree evolution metrics of dataset citibike for year 2018.

For example, the ANNDE of  $v_1$  in the interval  $\tau = [\omega_1, \omega_5)$  is  $degevenn(v_1, \tau) = \{1, 1, 2.67, 2\}$ . For our small example graph, the ANND remains quite small in this interval, which means that the popularity of the neighbours of  $v_1$  does not increase much. Fig. 4a shows the resulting ANNDE time series of a selected rental station for the real world bike-sharing graph we are using in our evaluation in Sect. 6. One can see that the tendency that rentals happen between popular stations are high during the summer months.

### 3.3 Graph-centric temporal degree metrics

After looking at metrics for individual vertices of a graph, we now develop metrics that concern an entire graph. Several metrics have already been defined for aggregating all vertices of a static graph, such as the *minimum*, *maximum*, and *average degree*.

**Definition 9 (Min/max/avg degree of a graph [LJ21])** The *minimum*, *maximum*, and *average degree* of a static graph  $G$  are defined as the minimum, maximum, and average value of all vertex degrees  $deg(v)$  for all  $v \in V$ . It holds:

$$deg_{min}(G) := \min\{deg(v) | v \in V\}, \quad (7)$$

$$deg_{max}(G) := \max\{deg(v) | v \in V\}, \quad (8)$$

$$deg_{avg}(G) := \frac{1}{|V|} \sum_{v \in V} deg(v), \quad (9)$$

with  $deg_{min}(G) \leq deg_{avg}(G) \leq deg_{max}(G)$ .

For the example graph in Fig. 2, the minimum, maximum, and average degrees are  $deg_{min}(G) = 3$ ,  $deg_{max}(G) = 7$  and  $deg_{avg}(G) = 5.34$ .

With the evolution of a graph, any aggregated graph metric can change over time. We therefore define the *minimum*, *maximum*, and *average temporal degree* as an aggregated value of all vertices  $V(\omega)$  in a temporal graph at time  $\omega$ .

**Definition 10 (Min/max/avg temporal degree)** The *minimum, maximum and average temporal degree* of a temporal graph  $G$  are the minimum, maximum and average values of all temporal vertex degrees at time  $\omega$ . With  $V(\omega)$  as the set of vertices at time  $\omega$  it holds:

$$degt_{min}(G, \omega) := \min\{degt(v, \omega) | v \in V(\omega)\}, \quad (10)$$

$$degt_{max}(G, \omega) := \max\{degt(v, \omega) | v \in V(\omega)\}, \quad (11)$$

$$degt_{avg}(G, \omega) := \frac{1}{|V(\omega)|} \sum_{v \in V(\omega)} degt(v, \omega), \quad (12)$$

with  $degt_{min}(G, \omega) \leq degt_{avg}(G, \omega) \leq degt_{max}(G, \omega)$ .

For the example graph in Fig. 2, at time  $\omega_4$ , the aggregated degrees are  $degt_{min}(G, \omega_4) = 1$ ,  $degt_{max}(G, \omega_4) = 5$  and  $degt_{avg}(G, \omega_4) = 2.67$ .

**Degree range.** The minimum degree reveals the smallest set of connections of a graph's vertices, whereas the maximum degree gives a measure of the most connections an vertex has in the graph. The difference between the minimum and maximum degree of any vertex in a graph is called the *degree range* [LJ21]. It provides a measure of the heterogeneity (or gap) between the connectivity of the most and the least connected vertices in a graph [LJ21].

**Definition 11 (Degree range [LJ21])** The *degree range* of a static graph  $G = (V, E)$ , denoted as  $deg_r(G)$ , is the difference between the maximum and minimum degree:

$$deg_r(G) = deg_{max}(G) - deg_{min}(G). \quad (13)$$

From a static view on the example graph of Fig. 2, the degree range is  $deg_r(G) = 7 - 3 = 4$ , which suggests that it has a high inequality related to connectivity. Now considering a temporal graph, the *temporal degree range* provides information about the degree range of a graph at a specific point in time.

**Definition 12 (Temporal degree range)** The *temporal degree range*  $degt_r(G, \omega)$  of a temporal graph  $G$  at time  $\omega$  is defined as the difference between the maximum and minimum temporal degree:

$$degt_r(G, \omega) = degt_{max}(G, \omega) - degt_{min}(G, \omega). \quad (14)$$

With respect to the example graph from Fig. 2, the temporal degree range at time  $\omega_4$  is  $degt_r(G, \omega_4) = 5 - 1 = 4$ , which is equal to the static metric, while at times  $\omega_1$ ,  $\omega_6$  and  $\omega_{10}$ , the temporal degree range is  $degt_r(G, \omega_1) = degt_r(G, \omega_6) = degt_r(G, \omega_{10}) = 1$ . Thus, as the graph evolves, the degree range changes as well.

To obtain any changes of the degree range over a defined time interval, we introduce the *degree range evolution* that defines a series of temporal degree range values for all time points in a given interval.

**Definition 13 (Degree range evolution)** The *degree range evolution*  $degevr_r(G, \tau) := \{x_1, x_2, \dots, x_m\}$  of a temporal graph  $G$  is a time series of elements  $x_i := degtr(G, \omega)$ , with  $1 \leq i \leq m$  and  $m = \omega_{end} - \omega_{start}$ . Each  $x_i$  represents the temporal degree range at time  $\omega_j$ , i.e.,  $x_1$  at time point  $\omega_{start}$  and  $x_m$  at  $\omega_{end} - 1$ , for the interval  $\tau = [\omega_{start}, \omega_{end}]$ . The temporal degree range is a special case of the degree range evolution:  $degevr_r(G, \tau) := \{degtr_r(G, \omega)\}$ , with  $\tau = [\omega_i, \omega_{i+1})$  as an interval with a single time point.

For the example graph of Fig. 2, the degree range evolution for  $\tau = [\omega_0, \omega_7)$  is  $degevr_r(G, \tau) = \{0, 1, 2, 5, 4, 3, 1\}$ , which shows a changing gap of connectivity in this interval. Fig. 4a shows the time series of the degree range evolution for the real world bike sharing graph we are using in our evaluations. One can see that the value is below 2 over the whole year which indicates a low inequality of rentals between all rental stations.

**Degree variance.** Besides the simple metric of range, Snijders introduced the more complex metric called *degree variance* of a graph [Sn81], which involves its average degree to characterize the *heterogeneity* in connectivity across nodes. This metric reveals information about the spread of both well-connected and not so well-connected vertices in a graph. It is formally defined as follows:

**Definition 14 (Degree variance [LJ21])** The *degree variance*  $deg_v(G)$  of a graph  $G$  is defined as the sum of the square of the difference between each vertex degree  $deg(v)$  and the average degree of the graph  $deg_{avg}(G)$ , divided by the total number of vertices  $|V|$ :

$$deg_v(G) := \frac{\sum_i (deg(v) - deg_{avg}(G))^2}{|V|}. \quad (15)$$

This metric quantifies the extent to which there are differences in the connectivity of the vertices in a graph. High differences in connectivity mean high variance; if all node degrees are the same then the degree variance is zero. If the example graph in Fig. 2 is considered static it has a degree variance of  $deg_v(G) = 2.89$ .

For temporal graphs, the degree of vertices can change over time, and so can the average degree as well as the number of vertices. Therefore, we formally define the *temporal degree variance* as follows:

**Definition 15 (Temporal degree variance)** The *temporal degree variance*,  $degt_v(G, \omega)$ , of a temporal graph  $G$  is defined as the sum of the square of the difference between each temporal vertex degree  $degt(v, \omega)$  and the temporal average degree of the graph  $degt_{avg}(G, \omega)$  at time  $\omega$ , divided by the total number of vertices  $|V(\omega)|$  at that time:

$$degt_v(G, \omega) := \frac{\sum_i (degt(v, \omega) - degt_{avg}(G, \omega))^2}{|V(\omega)|}. \quad (16)$$

Considering the example graph in Fig. 2 at  $\omega_4$ , the temporal degree variance is  $degt_v(G, \omega_4) = 2.89$ , which is equal to the static value since the inequality of connectivity is the same for this small example. In contrast, at time  $\omega_1$ , the temporal degree variance is  $degt_v(G, \omega_1) = 0.22$  since there is a quite high equality of connectivity at this time. To evaluate whether and how the degree variance changes in a given time interval, i.e., if the inequality of degrees in a graph decreases or increases over time, or if it retains a similar value, we define the *degree variance evolution*.

**Definition 16 (Degree variance evolution)** *The degree variance evolution  $degev_v(G, \tau) := \{x_1, x_2, \dots, x_m\}$  of a temporal graph  $G$  is a time series of elements  $x_i := degt_v(G, \omega)$ , with  $1 \leq i \leq m$  and  $m = \omega_{end} - \omega_{start}$ . Each  $x_i$  represents the temporal degree variance at time  $\omega_j$ , i.e.,  $x_1$  at time point  $\omega_{start}$  and  $x_m$  at  $\omega_{end} - 1$ , for the interval  $\tau = [\omega_{start}, \omega_{end})$ . Further, the temporal degree variance is a special case of the degree variance evolution:  $degev_v(G, \tau) = \{degt_v(G, \omega_i)\}$  with  $\tau = [\omega_i, \omega_{i+1})$  as an interval with a single time point.*

The degree variance evolution of vertex  $v_1$  in the example graph of Fig. 2, for time interval  $\tau = [\omega_0, \omega_5)$ , is the series:  $degev_v(G, \tau) = \{0, 0.22, 0.89, 2.89, 2.89\}$ . The degree variance increases over time in this example, which indicates a growth of the inequality of the vertex' connectivity. With regard to the real world bike sharing graph, Fig. 4b shows the degree variance evolution of the temporal graph. The inequality of the rental stations' utilization is low over the whole year but reaches its lowest values in the winter months.

## 4 Degree evolution algorithm

We now describe a baseline algorithm that calculates the degree evolution (see Definition 4) for all vertices in a temporal graph.

We assume that the input is a temporal graph  $G = (V, E)$  including a set of temporal vertices  $V$  and temporal edges  $E$  according to the TPGM model described in Sect. 3.1, where the degree type  $\Psi = \{in, out, both\}$  is given as configuration parameter. The output of the algorithm is a time series representing the degree evolution for each vertex, where we reduce the size of the result by merging succeeding time points without a degree change into intervals. These intervals are tuples  $\langle v_{id}, \tau_i, degt(v_{id}, \omega_j) \rangle$ , where  $v_{id}$  is a vertex identifier,  $\tau_i$  is the interval in which the degree is valid without interruption, and  $degt(v_{id}, \omega_i)$  the constant temporal degree of  $v_{id}$  for any time point  $\omega_j$  of the interval  $\tau_i$ . We split the algorithm into five steps which we described next.

**(1) Vertex mapping.** For each vertex  $v \in V$  we extract the vertex identifier and its time interval into a tuple  $\langle v_{id}, \omega_{start}, \omega_{end} \rangle$ . This tuple is later used as input of step (5). This step can be skipped if the vertex times are not of relevance. Considering our example graph

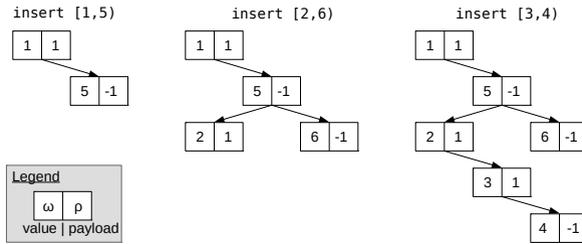


Fig. 5: Degree tree building for vertex  $v_1$  and  $\Psi = out$ .

in Fig. 2, each of the graph’s vertices  $V = \{v_1, v_2, v_3\}$  is mapped to a tuple, resulting in a set of three tuples  $\langle v_1, 0, \infty \rangle, \langle v_2, -\infty, \infty \rangle$  and  $\langle v_3, 0, 11 \rangle^7$ .

(2) **Edge mapping.** For each edge  $e \in E$  we extract the required vertex identifiers and the edge’s time interval into one or two tuples  $\langle v_{id}, \omega_{start}, \omega_{end} \rangle$  depending on the degree type  $\Psi$ . For  $\Psi = in$ , one tuple is created with  $v_{id} \leftarrow t_{id}$  (the target vertex identifier), for  $\Psi = out$  one tuple is created with  $v_{id} \leftarrow s_{id}$  (the source vertex identifier), and for  $\Psi = both$  both of these tuples are created. Considering the example graph in Fig. 2 and  $\Psi = out$ , each of the graphs edges  $E = \{e_1, e_2, \dots, e_8\}$  is mapped to one tuple as described above. For example, edge  $e_4$  is mapped to  $\langle v_2, 6, 10 \rangle$ , whereas  $e_5$  is mapped to  $\langle v_2, 3, 6 \rangle$ .

(3) **Interval collection.** We group the set of tuples  $\langle v_{id}, \omega_{start}, \omega_{end} \rangle$  from step (2) by vertex identifier and create a mapping  $v_{id} \rightarrow I_{v_{id}} = \{\tau_0, \tau_1, \dots, \tau_n\}$  which assigns a unsorted set of edge intervals  $I_{v_{id}}$  to the corresponding vertex identifier. For vertex  $v_1$  and  $\Psi = out$  of our example, the mapping to the collection of all incident edge intervals is  $v_1 \rightarrow I_{v_1} = \{[1, 5), [2, 6), [3, 4)\}$ .

(4) **Capture degree evolution.** For each vertex  $v$  and its corresponding unsorted set of (incoming, outgoing, or both) edge intervals created in step (3), a data structure maintaining the rise or fall of the metric at all respective points in time, i.e., when the degree of the vertex changes, is needed. A baseline approach is the maintenance of a typed list holding two types of points in time: the lower interval bounds which indicate a degree rise of 1, and the upper interval bounds which indicate a fall of 1. The space complexity is always  $O(n)$  with  $n = 2 \cdot |I_{v_{id}}|$ , i.e., the number of all time points including duplicates. All points in time can be inserted with a time complexity  $O(n)$  ( $O(1)$  each), and the list has to be sorted before the iteration which costs  $O(n \cdot \log(n))$ . The degree evolution for this vertex can be created by iterating the list (with  $O(n)$ ) and adding 1 to a aggregate value for all lower interval bounds and -1 for all upper bounds.

An alternative is a Binary Search Tree (BST) [Be75]  $T_v$ . Each node of the tree has a value  $\omega \in \Omega$  and a payload  $\rho \in \mathbb{Z}$ .  $\omega$  represents a point in time, whereas  $\rho$  (initialized with 0) stores an aggregated value indicating the quantity of change (positive or negative) of the

<sup>7</sup>Note that we use integers for time points to improve readability.

degree at this specific time  $\omega$  compared to the aggregated value of the evolution until this point in time. For a left-close right-open interval  $\tau = [\omega_{start}, \omega_{end})$ , the payload  $\rho$  of node  $\omega_{start}$  is increased by 1, whereas  $\rho$  of  $\omega_{end}$  is decreased by 1. Further, the left child node  $\omega_l$  of a parent node  $\omega_p$  has a value  $\omega_l < \omega_p$  and the right child node  $\omega_r$  has a value  $\omega_r > \omega_p$ , respectively. The worst case space complexity is  $O(n)$ , too, but having  $n$  without duplicate time points. The time complexity of inserting a node in this tree is  $O(\log(n))$  on average ( $O(n)$  if all time points are different). The random insertion of points in time, while keeping the tree sorted, and the lower memory requirements by avoiding duplicated points in time, is our reason for choosing the BST, which will be called *degree tree* in the following. Thus, the output of this step (4) is a mapping  $v_{id} \rightarrow T_v$  that assigns a degree tree to its corresponding vertex identifier.

If we again consider  $v_1$  in our example, the building of the degree tree  $T_{v_1}$  assuming  $\Psi = out$  is shown in Fig. 5. Inserting the interval  $[1, 5)$  first inserts a node with value  $\omega = 1$  and payload  $\rho = 1$ , and then a node with  $\omega = 5$  and payload  $\rho = -1$ . For the subsequent two intervals, four additional nodes are added. A degree tree with six nodes is the result, as shown on the right side.

**(5) Tree traversal and result collection** For each vertex, we now have a degree tree  $T_v$  that represents the degree evolution of this vertex for the degree type  $\Psi$ , and the lower and upper bounds of the vertex' validity interval,  $\omega_{start}$  and  $\omega_{end}$ . If the validity of the vertices can be neglected, a default minimum and maximum time point can be used as initial values. Each degree tree is now traversed using Depth First Search (DFS) [Ta72] and in-order traversal (LNR) starting at the root node to obtain an ascending order of points in time. Algorithm 1 outlines this step.

The algorithm starts by traversing the tree  $T_v$  in line 5 with the recursive function `IN-ORDERDFS` (lines 8 to 11). Function `PROCESSNODE` describes the logic of a node visit, where we first handle the special case of an vertex lower interval bound that is equal to the value of first visited node of the tree (lines 13 to 15). For every following visited node, the resulting temporal degree tuple is collected in line 17 if payload  $\rho \neq 0$ .

Next, to get the degree for the subsequent interval, the payload  $\rho$  is first added to  $d$  (line 18), and second the time point  $\omega$  is remembered as lower interval bound for the next interval (line 19). After all nodes of the tree are visited, we check for a remaining time interval from the last time point  $\omega_{last}$  to the vertex upper interval bound  $\omega_{max}$  and collect a last tuple with  $d = 0$  accordingly (line 7). The final algorithm output is a series of tuples  $\langle v_{id}, \tau, deg_t(v_{id}, \omega) \rangle$ , with  $deg_t(v_{id}, \omega)$  as constant temporal degree for all time points  $\omega \in \tau$ , that were collected by both `collect()` calls (lines 7 and 17).

For a better understanding, we exemplary go through Algorithm 1 by using the degree tree  $T_{v_1}$  of vertex  $v_1$ , shown on the right side in Figure 5, as input. Remember this is the representation of the outdegree of  $v_1$ . In addition, from step (1), the algorithm gets the lower bound  $\omega_{start} = 0$  and upper bound  $\omega_{end} = \infty$  of the vertex interval as input

**Algorithm 1:** Tree traversal and result collection

---

```

Data:  $T_v, v_{id}, \omega_{start}, \omega_{end};$  /* Input data */
1  $\omega_{last} \leftarrow \omega_{start};$  /*  $\omega_{start} = -\infty$  if not given */
2  $\omega_{max} \leftarrow \omega_{end};$  /*  $\omega_{end} = \infty$  if not given */
3  $d \leftarrow 0;$  /* Initialize degree with 0 */
4 Function Main():
5   InOrderDFS( $T_v$ ); /* Traverse the tree with in-order DFS */
6   if  $\omega_{last} < \omega_{max}$  then /* Check for last remaining interval */
7      $collect(\langle v_{id}, [\omega_{last}, \omega_{max}], d \rangle);$  /* Collect tuple for last interval */
8 Function InOrderDFS( $tree$ ):
9   if  $tree.left \neq null$  then InOrderDFS( $tree.left$ );
10  ProcessNode( $tree.value, tree.payload$ );
11  if  $tree.right \neq null$  then InOrderDFS( $tree.right$ );
12 Function ProcessNode( $\omega, \rho$ ):
13  if  $\omega_{last} == \omega$  then /* Check first node visit */
14     $d \leftarrow d + \rho;$  /* Add payload to degree */
15    return; /* Leave function */
16  if  $\rho \neq 0$  then /* Check if the degree changes */
17     $collect(\langle v_{id}, [\omega_{last}, \omega], d \rangle);$  /* Collect tuple */
18     $d \leftarrow d + \rho;$  /* Add degree change to degree */
19     $\omega_{last} \leftarrow \omega;$  /* Remember  $\omega$  for next call */

```

---

parameters to initialize  $\omega_{last}$  and  $\omega_{max}$ . During the in-order traversal of the DFS, function ProcessNode( $\omega, \rho$ ) is called first with the arguments (1, 1) (value.payload), followed by (2, 1), (3, 1), (4, -1), (5, -1) and (6, -1).

According to the first tuple, the interval  $[0, 1)$  is defined and collected as part of the first resulting temporal degree tuple  $\langle v_1, [0, 1), 0 \rangle$  afterwards (line 17). Then, the payload 1 is added to the degree value  $d$  (line 18) and the timestamp value 1 is remembered in variable  $\omega_{last}$  (line 19). In the next function call with input tuple (2, 1), an interval  $\tau \leftarrow [1, 2)$  is defined and collected together with the current degree value of  $d$  which is 1. The collected result tuple is thus  $\langle v_1, [1, 2), 1 \rangle$ . Again, the degree value is updated by the payload and the timestamp is remembered. For the remaining four input tuples (3, 1), (4, -1), (5, -1) and (6, -1) will be the following result tuples collected:  $\langle v_1, [2, 3), 2 \rangle$ ,  $\langle v_1, [3, 4), 3 \rangle$ ,  $\langle v_1, [4, 5), 2 \rangle$  and  $\langle v_1, [5, 6), 1 \rangle$ .

To collect also the remaining interval from 6 to  $\infty$ , the condition (line 7) checks whether the largest timestamp in the tree ( $\omega_{last}$ ) is smaller than the maximum timestamp ( $\omega_{max} = \infty$ ). Since this is true in our case, we define the remaining interval  $\tau = [6, \infty)$  and collect the output tuple  $\langle v_1, [6, \infty), 0 \rangle$  which states that the degree of  $v_1$  is 0 for the interval  $[6, \infty)$ . The result of this final step is a compact representation of the *degree evolution* of the outdegree of vertex  $v_1$  as defined by Definition 4:  $dege v^+(v_1, [0, \infty)) = \{\langle 0, [0, 1) \rangle, \langle 1, [1, 2) \rangle, \langle 2, [2, 3) \rangle, \langle 3, [3, 4) \rangle, \langle 2, [4, 5) \rangle, \langle 1, [5, 6) \rangle, \langle 0, [6, \infty) \rangle\}$ .

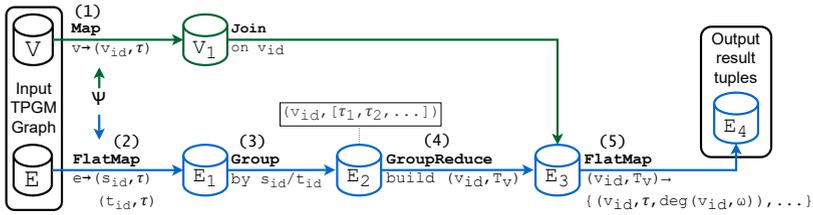


Fig. 6: Implementation details of the Degree Evolution-Operator.

## 5 Distributed implementation

The ability to process very large graphs efficiently is often a limitation of existing graph processing systems [Sa20], requiring partitioning of large graphs and distributed processing for example of analytical tasks. There are distributed graph processing systems, such as Tegra [Iy21] based on Apache Spark [Za16], or GRADOOP [Ro22, Ro21] which uses Apache Flink [Ca15]. An analytical operator in GRADOOP is a smart combination of Flink transformations. A Flink transformation, e.g., *map*, *flatMap* and *join*, is a processing unit that can be applied in parallel on a distributed Flink DataSet. A DataSet represents a distributed collection of elements of the same type in Apache Flink. Its tuples are distributed among all nodes of a cluster according to a partitioning strategy. We use this operator concept for our distributed implementation of the algorithm described in Sect. 4.

Fig. 6 shows an architectural sketch of a *Degree Evolution-Operator*<sup>8</sup> as a Directed Acyclic Graph (DAG) representing multiple Flink transformations that are applied on the input graph DataSets:  $V$  with  $v_i = \langle v_{id}, \tau \rangle$  and  $E$  with  $e_i = \langle e_{id}, s_{id}, t_{id}, \tau \rangle$ . The enumeration of the data flow follows the algorithm steps given in Sect. 4.

First, in step (1), each vertex of the input vertex DataSet  $V$ , is mapped to a minimal representation holding the vertex identifier and the bounds of the vertex’ time interval. The resulting DataSet is named  $V_1$  in the figure. If the temporal information of the vertices can be neglected, this step can be skipped and default min/max timestamps can be used as input to step (5), which avoids the later described distributed join. Then, we apply a *FlatMap* transformation, step (2), to the edge DataSet  $E$  that is configured by the degree type ( $\Psi \in \{in, out, both\}$ ) as selected by the user. According to the degree type, one or two tuples of the format  $\langle v_{id}, \omega_{start}, \omega_{end} \rangle$  are extracted from an input edge tuple (step (2) in Sect. 4). The resulting DataSet is denoted as  $E_1$ .

On  $E_1$ , we apply a *Group* transformation which groups all entities by the vertex identifier, and creates a set of tuples  $\langle \omega_{start}, \omega_{end} \rangle$  for each group. In the figure, this step is marked by (3), whereas the resulting grouped DataSet is denoted as  $E_2$ . Due to the grouping,  $E_2$  is partitioned by the vertex identifier. For each group, we apply a *GroupReduce* transformation

<sup>8</sup>The operator code is open-source: <https://github.com/dbs-leipzig/gradoop/tree/develop/gradoop-temporal/src/main/java/org/gradoop/temporal/model/impl/operators/metric>.

|                      | $ V $   | $ E $   | Size (GB) | $\sum  degev() $ |
|----------------------|---------|---------|-----------|------------------|
| <b>LDBC SF1</b>      | 3.2 M   | 17.3 M  | 4.2       | 30.6 M           |
| <b>LDBC SF10</b>     | 30.0 M  | 176.6 M | 42.3      | 319.6 M          |
| <b>LDBC SF100</b>    | 282.6 M | 1.77 B  | 421.9     | 3.18 B           |
| <b>Citi Bike</b>     | 1174    | 97.5 M  | 22.6      | 381.0 M          |
| <b>Stackoverflow</b> | 462.9 M | 664.8 M | 199.0     | 1.3 B            |

Tab. 1: Dataset statistics, including their sizes on HDFS and number of result set tuples for  $\Psi = both$ , i. e.,  $\sum_{i=1}^{|V|} |degev(v_i)|$ . For example, 3.18B tuples result for the LDBC dataset with SF 100.

in step (4) which calls a user-defined function for each group. This function receives the whole group at once and produces a mapping  $v_{id} \rightarrow T_v$  assigning a degree tree to its corresponding vertex identifier, represented as a tuple  $\langle v_{id}, T_v \rangle$ . The resulting tuples are part of DataSet  $E_3$ , which is partitioned by the vertex identifier.

Now, each tuple of  $V_1$  needs to be joined by the vertex identifier to its corresponding degree tree tuple of DataSet  $E_3$  to extend it with the interval bounds of the vertex. As said before, this step can be optionally skipped. As a result of the join, the DataSet  $E_3$  consists of tuples  $\langle v_{id}, T_v, \omega_{start}, \omega_{end} \rangle$ . As a last step, annotated with a (5), a *FlatMap* transformation is applied on DataSet  $E_3$  where its internal logic implements the tree traversal and result collection process defined in Algorithm 1. For each input tuple, the transformation produces multiple (at least one) result tuples in the form  $\langle v_{id}, \tau, deg(v_{id}, \omega) \rangle$ , describing the constant temporal degree (see Definition 3) of vertex  $v \in V$  (identified by  $v_{id}$ ) for the whole interval  $\tau$ . The resulting DataSet is named  $E_4$ .

## 6 Experimental Evaluation

We now evaluate the runtime and scalability of the temporal degree operator we discussed in Sect. 5 with respect to increasing data set and cluster sizes. We ran all experiments on a cluster with 16 worker nodes connected via 1 GBit Ethernet, where each worker consists of a E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM, two 4 TB SATA disks, and running openSuse 13.2, Hadoop 2.7.3 and Flink 1.9.0. On a worker node, a Flink Task Manager [Ca15] is configured with 6 task slots and 40GB memory.

We use three datasets for the evaluation, referred to as *LDBC* [Io16] (a synthetic social network in three scale factors), *citibike*<sup>9</sup> and *stackoverflow*<sup>10</sup> (both real-world data). In Fig. 4 we show example time series of four evolution metrics for the citibike dataset. Each graph is stored distributed using the Hadoop Distributed File System (HDFS) by hash partitioning as two datasets  $V$  and  $E$ . Table 1 shows statistics of the three datasets with the different scaling factors (SF) for LDBC. Each experiment includes reading the graph dataset from

<sup>9</sup><https://www.citibikenyc.com/system-data/> (visited 2022-10-01).

<sup>10</sup><https://archive.org/details/stackexchange> (visited 2022-10-01).

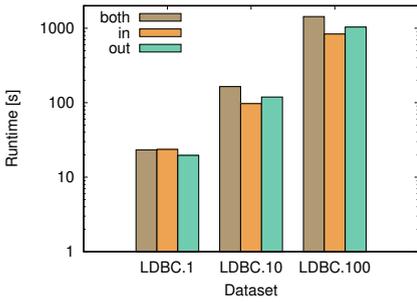


Fig. 7: Runtimes for linearly growing dataset sizes.

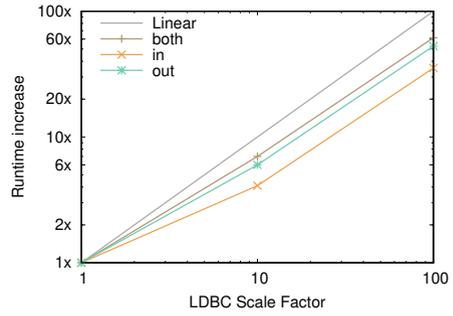


Fig. 8: Factor of runtime increase for linearly growing dataset sizes.

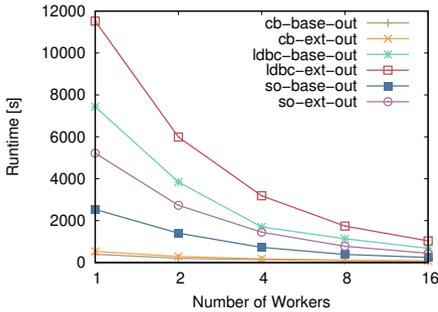


Fig. 9: Runtimes for #workers with  $\Psi = out$ .

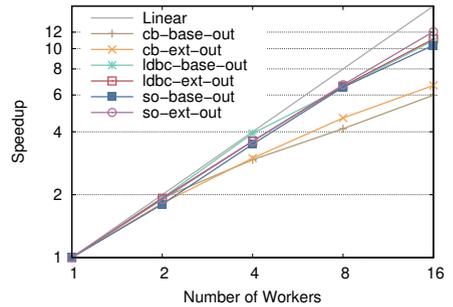


Fig. 10: Speedup of algorithm for  $\Psi = out$ .

the HDFS, executing the specific workflow, and finally writing all results back to the HDFS. We ran each experiment five times and report average runtimes.

**Impact of dataset size.** Fig. 7 and 8 show the impact of the dataset size to the operator runtime with full parallelism of 16 workers with respect to different degree types  $\Psi \in \{in, out, both\}$ . While Fig. 7 shows the actual runtime in seconds for all three dataset sizes, Fig. 8 visualizes the factor by which the runtime has increased compared to the runtime of the LDBC SF1 dataset. For example, the runtime for the LDBC SF1 dataset for  $\Psi = both$  is only 23.3 seconds, for LDBC SF10 164.6 seconds (factor 7 higher compared to LDBC SF1) and for LDBC SF100 1433.6 seconds (factor 61). The best result is given by degree type  $\Psi = in$ , where the runtimes of LDBC SF100 are only 35.4 times larger compared to LDBC SF1, although the dataset is 100 times larger. From LDBC SF10 to LDBC SF100 the runtimes of all three degree types rise equally.

The results, specifically Fig. 8, show that a linear increase of the dataset size leads to only a sublinear increase in the running time for a constant graph structure. Further, the runtimes

of  $\Psi = \textit{both}$  are always higher compared to the others which is due to the double amount of collected tuples in step (2), as we discussed in Sect. 5.

**Impact of worker count.** We next examine the runtime and scalability of the algorithm for all datasets. In addition, the effect of excluding the vertex time information as described in Sect. 4 is evaluated. Without using the vertex time, the complete step (1) and the expensive join after step (4) can be avoided (see Sect. 5). In the following, we refer to an execution without vertex time as *base* and *extended* for the full algorithm. The results in Fig. 9 show that the mentioned higher complexity has a significant impact on the running time. For example, the runtime on a single machine for the citibike dataset is 397.6 seconds (base) and 533.6 seconds (extended), which means an increase of 34.2%. For the stackoverflow dataset, the execution takes 2,536 seconds (base) and 5,216 seconds (extended), which means almost doubling the runtime on a single machine.

The more workers are added, the smaller the runtime and the difference between the two algorithm variants, which can be seen in Fig. 10. With the citibike dataset, we can see that the runtimes on a single machine are already low and that only a moderate improvement can be achieved through horizontal scaling of resources. For this dataset, we reach a speedup of about 6.7 for 16 machines using the extended variant, while for the LDBC SF100 and stackoverflow datasets we achieve a speedup of up-to 11.1 and 12.07, respectively.

## 7 Conclusion

Most graphs that model real-world entities and their relationships are dynamic, where edges and vertices can be valid for only a certain period of time. One simple but often used centrality measure is the degree centrality using a vertex' degree to judge it's popularity in a network. We show in this work that it is necessary to determine a vertex degree over time, to know exactly *when* a node has which degree and *how long* this value is valid and in which quantity it does change over time. We therefore provide temporal extensions to the vertex degree metric itself, its aggregations and others based on it, namely the degree range, the degree variance and the ANND, and define them formally. We further describe an algorithm to calculate the newly introduced degree evolution for all vertices of a temporal graph. We implemented the algorithm as a graph analysis operator in GRADOOP [Ro22], an open-source distributed graph analysis system.

We evaluated runtimes and scalability of the operator on a cluster with 16 machines to determine the impact of different datasets and sizes. In summary, we have shown that a linear increase in the dataset size leads to only a sublinear increase in runtime of our algorithm. We also showed that the operator scales well by increasing the number of machines. Speedup values between 10 and 12 were achieved on 16 machines using the two largest datasets.

**Acknowledgement.** The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany and by the Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus for ScaDS.AI.

## Bibliography

- [Be75] Bentley, Jon Louis: Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, sep 1975.
- [Ca06] Capocci, Andrea et al.: Preferential attachment in the growth of social networks: The internet encyclopedia Wikipedia. *Physical review E*, 74(3):036116, 2006.
- [Ca15] Carbone, Paris et al.: Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.
- [Ca21] Casteigts, Arnaud; Meeks, Kitty; Mertzios, George B.; Niedermeier, Rolf: Temporal Graphs: Structure, Algorithms, Applications (Dagstuhl Seminar 21171). *Dagstuhl Reports*, 11(3):16–46, 2021.
- [Ci20] Ciaperoni, Martino; Galimberti, Edoardo; Bonchi, Francesco; Cattuto, Ciro; Gullo, Francesco; Barrat, Alain: Relevance of temporal cores for epidemic spread in temporal networks. *Scientific reports*, 10(1):1–15, 2020.
- [Di10] Diestel, Reinhard: *Graph Theory*, 4th Edition. Springer, 2010.
- [Fr78] Freeman, Linton C: Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.
- [GS20] Gandhi, Swapnil; Simmhan, Yogesh: An interval-centric model for distributed computing over temporal graphs. In: *IEEE 36th International Conference on Data Engineering (ICDE)*. pp. 1129–1140, 2020.
- [GY03] Gross, Jonathan L; Yellen, Jay: *Handbook of graph theory*. CRC press, 2003.
- [Ho18] Holme, Petter: Temporal Networks. In: *Encyclopedia of Social Network Analysis and Mining*. 2nd Ed. Springer, 2018.
- [HR21] Halawa, Hassan; Ripeanu, Matei: Position paper: bitemporal dynamic graph analytics. In: *GRADES NDA*. pp. 1–12, 2021.
- [Io16] Iosup, Alexandru et al.: LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *Proc. of the VLDB Endow.*, 9(13):1317–1328, 2016.
- [Iy21] Iyer, Anand Padmanabha; Pu, Qifan; Patel, Kishan; Gonzalez, Joseph E; Stoica, Ion: TEGRA: Efficient Ad-Hoc Analytics on Evolving Graphs. In: *NSDI*. pp. 337–355, 2021.
- [JNB03] Jeong, Hawoong; Néda, Zoltan; Barabási, Albert-László: Measuring preferential attachment in evolving networks. *EPL (Europhysics Letters)*, 61(4):567, 2003.
- [KA12] Kim, Hyoungshick; Anderson, Ross: Temporal node centrality in complex networks. *Physical Review E*, 85(2):026107, 2012.
- [Ko09] Kostakos, Vassilis: Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023, 2009.
- [Li15] Li, Yexin; Zheng, Yu; Zhang, Huichu; Chen, Lei: Traffic prediction in a bike-sharing system. In: *Proc. of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. pp. 1–10, 2015.

- [LJ21] Lizardo, Omar; Jilbert, Isaac: , Graph Metrics. <http://olizardo.bo1.ucla.edu/classes/soc-111/lessons-winter-2022/4-lesson-graph-metrics.html>, 2021. [Online; accessed 2022-10-01].
- [Lo20] Long, Li; Abbas\*, Khushnood; Ling\*, Niu; Jafar Abbas, Syed: Ranking Nodes in Temporal Networks: Eigen Value and Node Degree Growth based. In: 2nd International Conference on Image Processing and Machine Vision. pp. 146–153, 2020.
- [Ne01] Newman, Mark EJ: Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001.
- [Ni13] Nicosia, Vincenzo; Tang, John; Mascolo, Cecilia; Musolesi, Mirco; Russo, Giovanni; Latora, Vito: Graph metrics for temporal networks. In: *Temporal Networks*, pp. 15–40. Springer, 2013.
- [RKC01] Riolo, Christopher S; Koopman, James S; Chick, Stephen E: Methods and measures for the description of epidemiologic contact networks. *J Urban Health*, 78(3):446–457, 2001.
- [Ro21] Rost, Christopher; Gómez, Kevin; Fritzsche, Philip; Thor, Andreas; Rahm, Erhard: Exploration and Analysis of Temporal Property Graphs. In: *EDBT*. pp. 682–685, 2021.
- [Ro22] Rost, Christopher; Gomez, Kevin; Täschner, Matthias; Fritzsche, Philip; Schons, Lucas; Christ, Lukas; Adameit, Timo; Junghanns, Martin; Rahm, Erhard: Distributed temporal graph analytics with GRADOOP. *The VLDB Journal*, 31:1–27, 2022.
- [Sa20] Sahu, Siddhartha; Mhedhbi, Amine; Salihoglu, Semih; Lin, Jimmy; Özsü, M Tamer: The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *The VLDB Journal*, 29(2):595–618, 2020.
- [SE20] Smith, Keith M; Escudero, Javier: Normalised degree variance. *Applied Network Science*, 5(1):1–22, 2020.
- [SK05] Saramäki, Jari; Kaski, Kimmo: Modelling development of epidemics with dynamic small-world networks. *Journal of Theoretical Biology*, 234(3):413–421, 2005.
- [Sn81] Snijders, Tom AB: The degree variance: an index of graph heterogeneity. *Social networks*, 3(3):163–174, 1981.
- [Ta72] Tarjan, Robert: Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- [TBF17] Thompson, William Hedley; Brantefors, Per; Fransson, Peter: From static to temporal network theory: Applications to functional brain connectivity. *Network Neuroscience*, 1(2):69–99, 2017.
- [TI20] Tlebaldinova, Aizhan; Nugumanova, Aliya; Baiburin, Yerzhan; Zhantassova, Zheniskul; Karmenova, Markhaba; Ivanov, Andrey: Temporal Network Approach to Explore Bike Sharing Usage Patterns. In: *VEHITS*. pp. 129–136, 2020.
- [Wa17] Wang, Zhiqiang; Pei, Xubin; Wang, Yanbo; Yao, Yiyang: Ranking the key nodes with temporal degree deviation centrality on complex networks. In: 2017 29th Chinese Control And Decision Conference (CCDC). IEEE, pp. 1484–1489, 2017.

- [Wu14] Wu, Huanhuan; Cheng, James; Huang, Silu; Ke, Yiping; Lu, Yi; Xu, Yanyan: Path problems in temporal graphs. Proc. of the VLDB Endowment, 7(9):721–732, 2014.
- [YvdHL17] Yao, Dong; van der Hoorn, Pim; Litvak, Nelly: Average nearest neighbor degrees in scale-free networks. arXiv preprint arXiv:1704.05707, 2017.
- [Za16] Zaharia, Matei et al.: Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM, 59(11):56–65, 2016.



## Session 5



# Discovering Multi-Dimensional Subsequence Queries from Traces – From Theory to Practice

Sarah Kleest-Meißner,<sup>1</sup> Rebecca Sattler,<sup>2</sup> Markus L. Schmid,<sup>3</sup> Nicole Schweikardt,<sup>4</sup>  
Matthias Weidlich<sup>5</sup>

**Abstract:** Subsequence-queries with wildcards and gap-size constraints (swg-queries, for short) are an expressive model for sequence data, in which queries are described by patterns over an alphabet of variables and types, along with a global window size and a number of gap-size constraints. They are evaluated over a trace, i.e., a sequence of types, by replacing variables by single types, while satisfying the window and the gap-size constraints. Kleest-Meißner et al. (Proc. ICDT 2022) formalised the task of discovering an swg-query that describes best a given sample consisting of a finite number of traces, and developed a discovery algorithm solving this task. However, in practical application scenarios, traces are often multi-dimensional, i.e., a trace corresponds to a sequence of tuples of types, which renders the existing technique inapplicable.

In this paper, we lift the notion of swg-queries to such a multi-dimensional setting, thereby enlarging the applicability of the query model and the techniques for query discovery. We introduce a mapping between one-dimensional and multi-dimensional sequence data, such that a multi-dimensional trace matches a multi-dimensional query if and only if the corresponding one-dimensional trace matches the corresponding one-dimensional query. We complement our formal results with a description of our prototypical implementation of query discovery for multi-dimensional sequence data. Results from evaluation experiments with real-world data indicate the feasibility of our approach.

**Keywords:** multi-dimensional subsequence queries on traces, detecting descriptive multi-dimensional queries, subsequences, embeddings

## 1 Introduction

Models for sequence data define an order for a set of data items [Bab+02], which typically follows from the order in which these items have been created, observed, or received. They

---

<sup>1</sup> Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany, kleemeis@informatik.hu-berlin.de. Supported by the German Research Foundation (DFG), CRC 1404: “FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis”

<sup>2</sup> Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany, rebecca.sattler@informatik.hu-berlin.de. Supported by the German Research Foundation (DFG), CRC 1404: “FONDA: Foundation of Workflows for Large-Scale Scientific Data Analysis”

<sup>3</sup> Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany, MLSchmid@MLSchmid.de. Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735)

<sup>4</sup> Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany, schweikn@informatik.hu-berlin.de

<sup>5</sup> Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany, matthias.weidlich@hu-berlin.de

facilitate the analysis of the evolution of some system over time and have been adopted in various domains. For instance, in cluster monitoring, sequence data describes the process of executing tasks on machines [Ver+15]; in urban transportation, sequence data captures the routes taken by vehicles [Art+14]; and in finance, sequence data represents a transaction history [TRP12].

Sequence data may be queried for relevant patterns by specifying which data items, in which order, and in which temporal context are of interest for a specific analysis question [Gia+20; CM12]. This way, *situations of interest* that happened in the past and materialized in historic data can be detected by evaluating a suitable subsequence query. Employing such a query over a stream of data items also enables the detection of such a situation immediately upon its occurrence, thereby enabling reactive and even pro-active applications.

As an example, consider a cluster monitoring scenario as illustrated in Figure 1. Here, data items indicate transitions in the lifecycle of a task, see Figure 1a. A query may then specify a subsequence of abnormal task execution in the cluster, e.g., as a sequence of data items that indicate that a task was scheduled, killed, and, after being treated in the same way twice (e.g., being updated twice), scheduled again for execution. Here, the respective subsequence is not necessarily continuous and certain lifecycle transitions that are not indicative may occur between the relevant ones. Figure 1b shows how such a query would be written following common languages for complex event recognition [Gia+20].

In practice, finding a suitable query that detects a particular situation is far from trivial, though. Users often know the time at which a situation occurred in the past, but lack insights into the exact materialization of the situation in the sequence data. To provide guidance in the formulation of an adequate query, it was therefore suggested to discover queries that describe patterns linked to the situation of interest [GCW16; MCT14]. These queries may then be interpreted and validated by a user in order to provide traceability and avoid overfitting.

Previously, we proposed a query language for describing subsequence queries with wildcards, a window size, and gap-size constraints [Kle+22], referred to as *swg-queries*. In essence, an *swg-query* defines a pattern over an alphabet of variables and types, a global window size, and gap-size constraints that bound the number of items that may occur between the queried types and variables. Taking up the query from Figure 1b, the respective *swg-query* includes: (i) a pattern SCH KIL  $x$   $x$  SCH with  $x$  denoting a variable; (ii) a global window size of at most 15 items; and (iii) gap-size constraints, e.g., (0, 10) to define that between zero and ten data items may occur between the type KIL and the first occurrence of variable  $x$ .

The general concept of subsequences has extensively been studied both in a purely combinatorial sense (in formal language theory, logic and combinatorics on words) and algorithmically (in string algorithms and bioinformatics); see the introductions of the recent papers [Gaw+21; Day+21] for a comprehensive list of relevant pointers. The problem of matching subsequences with gap-constraints (and analysis problems with respect to the set

(SUB, SCH, EVI, SCH, KIL, UPD, CHE, UPD, SCH, FIN)

(a) Sequence of data items recorded for a task and indicating the task’s lifecycle: Submitted (SUB), scheduled (SCH), evicted (EVI), killed (KIL), updated (UPD), checked (CHE), finished (FIN).

```
PATTERN SEQ(Item a, Item b, Item c, Item d, Item e)
WHERE a.status = e.status = SCH AND b.status = KIL AND c.status = d.status
WITHIN 15 data items
```

(b) A query over sequences of data items following common languages for complex event recognition [Gia+20]. It detects if a task was scheduled, killed, and, after treated in the same way twice (e.g. being updated twice), scheduled again for execution. It matches the sequence in Figure 1a.

Fig. 1: Illustration of a query over sequence data in the domain of cluster monitoring.

of all gap-constrained subsequences of given strings) has been investigated in the recent papers [Day+22; Kos+22a] (see also [Kos+22b] for a survey).

Patterns with variables were introduced by Angluin [Ang80]; they play a central role for inductive inference, in formal language theory and combinatorics on words (see [SA95; MS19; RS97]). Syntactically, our swg-queries are Angluin-style patterns, but adapted in a way that variables refer solely to single types (whereas in Angluin’s semantics they refer to finite sequences of types) and matches are further constrained by a global window size and a number of gap-size constraints (such constraints are not available in Angluin’s pattern queries).

Despite the fundamental semantic differences between swg-queries and Angluin-style patterns, it is possible to adapt concepts and algorithms from inductive inference of the so-called *pattern languages* that can be described by Angluin-style patterns. Most importantly, the classical concept of *descriptive patterns* (already introduced in [Ang80], see also [FR10; FR13]), can be adapted to swg-queries [Kle+22]: a query  $q$  is called *descriptive* for a given sample  $\mathcal{S}$  (i.e., a finite set of sequences of data items) and a given support threshold  $sp$  if it matches in at least a fraction of  $sp$  sequences of  $\mathcal{S}$  and there is no strictly more restrictive query  $q'$  that also matches in a fraction of at least  $sp$  sequences of  $\mathcal{S}$ .

For classical Angluin-style semantics, *Shinohara’s algorithm* [Shi82] computes a descriptive pattern query upon input of a sample  $\mathcal{S}$  and the support threshold  $sp = 1$  (see also [Fer+18] for a thorough analysis and extensions of Shinohara’s algorithm). In [Kle+22] we presented an adaptation and extension of Shinohara’s algorithm that is capable of discovering, upon input of a sample  $\mathcal{S}$  and a support threshold  $sp \leq 1$ , a descriptive swg-query — and different executions of this algorithm may be used to compute a number of different descriptive swg-queries.

However, a major drawback of swg-queries as well as related models of Angluin-style patterns is that they are based on a *one-dimensional* model of sequence data, i.e., data items refer to atomic types. As a consequence, they are not applicable in many practical scenarios in which sequence data comprise items that are instances of a *multi-dimensional* schema.

```
(job=1, task=2, machine=5, status=SCH, priority=low)
(job=4, task=3, machine=5, status=SCH, priority=high)
(job=2, task=1, machine=1, status=UPD, priority=high)
(job=1, task=2, machine=5, status=EVI, priority=low)
(job=1, task=2, machine=3, status=SCH, priority=low)
```

(a) A sequence of multi-dimensional data items. Each data item has five attributes that characterise the job to which a task belongs (*job*), the identifier of the task (*task*), the machine to which the task is assigned (*machine*), the task's lifecycle transition (*status*), and the execution priority of the task (*priority*).

```
PATTERN SEQ(Item a, Item b, Item c)
WHERE a.status = b.status = SCH AND c.status = EVI
AND a.job = c.job AND a.task = c.task AND
AND a.machine = b.machine AND b.priority = high
WITHIN 10 data items
```

(b) A query over sequences of multi-dimensional data items (again, following common languages for complex event recognition [Gia+20]). The query is matched in the sequence depicted in Figure 2a by associating a, b, and c with the first, second, and fourth tuple of the sequence.

Fig. 2: Illustration of a query over multi-dimensional sequence data.

Considering the application domain of cluster monitoring, data items may capture not only the lifecycle transitions related to a task, but also the job to which the task belongs, the assigned machine, and the priority of task execution, see Figure 2a. These attributes enable the definition of more elaborate subsequence queries that incorporate predicates over the respective values of data items. For instance, the query depicted in Figure 2b detects the situation that a task is scheduled on a machine for which, subsequently, the scheduling of a task of a high-priority job on the same machine leads to the eviction of the first task.

In this paper, we address the above limitation by lifting swg-queries to multi-dimensional sequence data. This way, we extend the applicability of the query model and also enable the discovery of descriptive queries from a sample database of multi-dimensional sequences. Our approach is to formulate a suitable mapping between one-dimensional and multi-dimensional sequence data that facilitates query evaluation: A multi-dimensional sequence matches a multi-dimensional query if, and only if, the corresponding one-dimensional sequence matches the corresponding one-dimensional query. We complement these formal contributions with a description of our prototypical implementation of query discovery for multi-dimensional sequence data. We further report on experiments on applying this prototype to a real-world dataset in the domain of cluster monitoring, i.e., the Google Cluster Traces [RWH11]. Our results indicate the general feasibility of discovering swg-queries from multi-dimensional sequence data and also shed light on the sensitivity of the runtime of the approach with respect to structural characteristics of the sequence database.

The rest of this paper is structured as follows. Section 2 introduces the multi-dimensional query model and relates it to the previously studied one-dimensional case. Section 3 provides a solution for the query discovery problem and briefly describes our implementation. Section 4 presents our experimental evaluation. Section 5 concludes the paper.

## 2 Multi-Dimensional Subsequence-Queries

This section introduces the syntax and semantics of *multi-dimensional subsequence-queries with wildcards and gap-size constraints*, for short: mswg-queries (Section 2.1). After briefly discussing their relation to the (one-dimensional) swg-queries introduced in [Kle+22] (Section 2.2) we present a way to encode mswg-queries as swg-queries (Section 2.3). Prior to this, we fix some basic notation.

Let  $\mathbb{N}$  and  $\mathbb{N}_{>1}$  be the set of non-negative integers and positive integers, respectively. For  $\ell \in \mathbb{N}$  we let  $[\ell] = \{i \in \mathbb{N} : 1 \leq i \leq \ell\}$ .

Let  $A$  be a non-empty set. We write  $A^*$  (and  $A^+$ ) for the set of all strings (and the set of all non-empty strings) over  $A$ . We denote the length of a string  $s$  by  $|s|$ . For a position  $i \in [|s|]$  we write  $s[i]$  to denote the letter at position  $i$  in  $s$ . A *factor* of a string  $s \in A^*$  is a string  $v \in A^*$  such that  $s = uvu'$  for  $u, u' \in A^*$ . A *subsequence* of a string  $t = t_1t_2 \cdots t_n$ , where  $t_i \in A$  for all  $i \in [n]$ , is a string  $s = s_1 \cdots s_m$  where  $m \leq n$  and there exist integers  $1 \leq i_1 < \cdots < i_m \leq n$  such that  $s_j = t_{i_j}$  for all  $j \in [m]$ ; the mapping  $e : [m] \rightarrow [n]$  with  $e(j) = i_j$  for all  $j \in [m]$  is called an *embedding* of  $s$  in  $t$ . For example, the string `acc` is a subsequence of the string `abaccb` with embedding  $e$  where  $e(1) \in \{1, 3\}$ ,  $e(2) = 4$  and  $e(3) = 5$ . We write  $s \preceq_e t$  to indicate that  $s$  is a subsequence of  $t$  with embedding  $e$ , and we suppress the subscript  $e$  if we only want to indicate that  $s$  is a subsequence of  $t$ .

For the rest of this paper we define  $\Gamma$  to be a (finite or infinite) alphabet with  $|\Gamma| \geq 2$ . The elements in  $\Gamma$  will be called *types*. Furthermore, we fix a countably infinite set  $\text{Vars}$  of *variables*, which is disjoint with the set  $\Gamma$  of types.

### 2.1 Syntax and Semantics of mswg-queries

We fix a number  $k \in \mathbb{N}_{>1}$  which we will henceforth call the *dimension*. We model a data item  $d$  with  $k$  attributes as an ordered  $k$ -tuple over  $\Gamma$ , i.e., an element in  $\Sigma := (\Gamma^k)$ .<sup>6</sup> A *k-dimensional trace over  $\Gamma$*  (for short: *k-trace*) is an element in  $\Sigma^+$ , i.e., a finite non-empty sequence of  $k$ -tuples over  $\Gamma$ . The *length*  $|t|$  of a  $k$ -trace  $t$  is the number of  $k$ -tuples it comprises — i.e.,  $|t|$  is  $t$ 's length as a string over alphabet  $\Sigma$ . We write  $\text{types}(t)$  for the set of types in  $\Gamma$  that occur in  $t$ .

**Example 1.** *We consider 5-dimensional data items with attributes `job`, `task`, `machine`, `status` and `priority` (which, for a unique tuple representation, are ordered as indicated above), and let  $\Gamma := \{\text{SCH}, \text{UPD}, \text{KIL}\} \cup \{0, 1, \dots, 10\}$ . For example, `(1, 2, 5, SCH, 0)` and*

<sup>6</sup> When considering  $(\Gamma^k)$  as an alphabet, i.e., each  $k$ -tuple is viewed as a single letter of this alphabet, we use brackets to visualize this.

$(1, 1, 5, \text{KIL}, 0)$  are two 5-dimensional data items (i.e., 5-tuples).  
 The following are two examples of 5-dimensional traces over  $\Gamma$ :

$$\begin{aligned} s &:= (1, 2, 5, \text{SCH}, 0) (1, 1, 5, \text{KIL}, 0) (1, 2, 5, \text{SCH}, 1) \\ t &:= (1, 2, 5, \text{SCH}, 0) (1, 2, 4, \text{UPD}, 0) (1, 1, 5, \text{KIL}, 0) (1, 2, 5, \text{SCH}, 1) (2, 3, 2, \text{UPD}, 1) \end{aligned}$$

These traces have length  $|s| = 3$  and  $|t| = 5$ . The mapping  $e : [3] \rightarrow [5]$  with  $e(1) = 1$ ,  $e(2) = 3$ , and  $e(3) = 4$  is an embedding of  $s$  in  $t$  and hence witnesses that  $s$  is a subsequence of  $t$ . From now on, we will illustrate such an embedding in the following way:

$$\begin{aligned} s &= (1, 2, 5, \text{SCH}, 0) && (1, 1, 5, \text{KIL}, 0) (1, 2, 5, \text{SCH}, 1) \\ t &= (1, 2, 5, \text{SCH}, 0) (1, 2, 4, \text{UPD}, 0) (1, 1, 5, \text{KIL}, 0) (1, 2, 5, \text{SCH}, 1) (2, 3, 2, \text{UPD}, 1) \end{aligned}$$

**Definition 2.** A  $k$ -dimensional subsequence-query with wildcards and gap-size constraints ( $k$ -swg-query, for short)  $q = (s, w, c)$  (over  $\text{Vars}$  and  $\Gamma$ ), consists of a query string  $s \in ((\text{Vars} \cup \Gamma)^k)^+$  (i.e.,  $s$  is a non-empty string of  $k$ -tuples built from variables and types), a global window size  $w \in \mathbb{N}_{\geq 1} \cup \{\infty\}$  with  $w \geq |s|$ , and a tuple of local gap-size constraints  $c = (c_1, c_2, \dots, c_{|s|-1})$ , where  $c_i = (c_i^-, c_i^+) \in \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ , such that  $c_i^- \leq c_i^+$  for every  $i \in [|s|-1]$  and  $|s| + \sum_{i=1}^{|s|-1} c_i^- \leq w$ .

We speak of *multi-dimensional subsequence-queries* (for short: *mswg-queries*) to refer to  $k$ -swg-queries for arbitrary dimension  $k \in \mathbb{N}_{\geq 1}$ . For an mswg-query  $q = (s, w, c)$  we write  $\text{types}(q)$  (or  $\text{types}(s)$ ) and  $\text{vars}(q)$  (or  $\text{vars}(s)$ ) to denote the set of types (from  $\Gamma$ ) and the set of variables (from  $\text{Vars}$ ), respectively, that occur in  $q$ 's query string  $s$ . Such a query  $q$  is called an  $(\ell, w, c)$ -query for  $\ell := |s|$ . We will refer to  $(\ell, w, c)$  as *query parameters*.

The semantics of mswg-queries is defined as follows: Each variable in a query string  $s$  serves as a *wildcard* representing an arbitrary type from  $\Gamma$ . A  $k$ -swg-query  $q = (s, w, c)$  *matches in a  $k$ -trace  $t$*  (in symbols:  $t \models q$ ), if the wildcards in  $s$  can be replaced by types in  $\Gamma$  in such a way that the resulting  $k$ -trace  $s'$  satisfies the following:  $t$  contains a factor  $t'$  of length at most  $w$  such that  $s'$  occurs as a subsequence in  $t'$  and for each  $i < \ell := |s|$  the gap between  $s'[i]$  and  $s'[i+1]$  in  $t'$  has length at least  $c_i^-$  and at most  $c_i^+$ . I.e.,  $t'$  is of the form  $s'[1] g_1 s'[2] g_2 \dots g_{\ell-1} s'[\ell]$  and  $c_i^- \leq |g_i| \leq c_i^+$  for all  $i \in [\ell-1]$ .

An alternative, more formal description of these semantics relies on the following additional notation: An embedding  $e : [\ell] \rightarrow [n]$  is said to *satisfy a global window size  $w$* , if  $e(\ell) - e(1) + 1 \leq w$ ; and we say  $e$  *satisfies a tuple  $c = (c_1, c_2, \dots, c_{\ell-1})$  of local gap-size constraints* (for  $\ell$  and  $w$ ), if  $c_i^- \leq e(i+1) - 1 - e(i) \leq c_i^+$  for all  $i < \ell$ . Consider a mapping  $\mu : (\text{Vars} \cup \Gamma) \rightarrow \Gamma$  with  $\mu(a) = a$  for all types  $a \in \Gamma$  (such mappings will henceforth be called *substitutions*). We lift  $\mu$  to a mapping from  $(\text{Vars} \cup \Gamma)^k$  to  $\Sigma := (\Gamma^k)$  by letting  $\mu((a_1, \dots, a_k)) := (\mu(a_1), \dots, \mu(a_k))$  for all  $(a_1, \dots, a_k) \in (\text{Vars} \cup \Gamma)^k$ ; and we further lift  $\mu$  to a mapping from query strings  $s \in ((\text{Vars} \cup \Gamma)^k)^+$  to  $k$ -traces of length  $\ell := |s|$  by letting  $\mu(s) = \mu(s[1]) \dots \mu(s[\ell])$ .

Using these notions, we obtain that a  $k$ -trace  $t$  matches a  $k$ -swg-query  $q = (s, w, c)$  if, and only if, there exist a substitution  $\mu : (\text{Vars} \cup \Gamma) \rightarrow \Gamma$  and an embedding  $e : [|s|] \rightarrow [|t|]$  such that  $\mu(s) \leq_e t$  and  $e$  satisfies  $w$  and  $c$ . We call  $(\mu, e)$  a witness for  $t \models q$ .

**Example 3.** We consider 5-dimensional data items with attributes *job*, *task*, *machine*, *status*, and *priority* (in this order) and types in  $\Gamma = \{\text{SCH, EVI, UPD, KIL}\} \cup \{0, 1, \dots, 10\} \cup \{h, l\}$  (with  $h, l$  being abbreviations for *high* and *low*). Consider the query from Figure 2b. This query searches for a subsequence of three 5-dimensional data items, the first two of which have status SCH (schedule) and the third of which has status EVI (evict) such that the following is true: the first and third data items are related to the same job and to the same task, the first and second data items are related to the same machine, and the second data item has a high priority; all within at most 10 data items.

This can be expressed as a 5-swg-query  $q = (s, w, c)$  as follows: The query string length is  $\ell := 3$ . The window size is  $w := 10$ . As there are no particular constraints on the gap sizes between the data items, the gap size constraints  $c$  are chosen to be  $c = ((0, \infty), (0, \infty))$  (meaning that each gap can be of arbitrary length). The query string  $s$  is

$$s := (x_j, x_t, x_m, \text{SCH}, y_1) (y_2, y_3, x_m, \text{SCH}, h) (x_j, x_t, y_4, \text{EVI}, y_5)$$

where  $x_j, x_t, x_m, y_1, \dots, y_5$  are pairwise distinct variables in  $\text{Vars}$ . The sequence of 5-dimensional data items depicted in Figure 2a corresponds to the 5-trace

$$t := (1, 2, 5, \text{SCH}, l) (4, 3, 5, \text{SCH}, h) (2, 1, 1, \text{UPD}, h) (1, 2, 5, \text{EVI}, l) (1, 2, 3, \text{SCH}, l).$$

Observe that  $t \models q$ , and a witness substitution  $\mu$  and embedding  $e$  can be illustrated as:

$$\begin{aligned} s &= (x_j, x_t, x_m, \text{SCH}, y_1) (y_2, y_3, x_m, \text{SCH}, h) && (x_j, x_t, y_4, \text{EVI}, y_5) \\ t &= (1, 2, 5, \text{SCH}, l) (4, 3, 5, \text{SCH}, h) (2, 1, 1, \text{UPD}, h) (1, 2, 5, \text{EVI}, l) (1, 2, 3, \text{SCH}, l) \end{aligned}$$

## 2.2 One-dimensional swg-queries

For the special case of dimension  $k = 1$  we identify  $k$ -tuples of elements in  $\text{Vars} \cup \Gamma$  with plain elements in  $\text{Vars} \cup \Gamma$ ; i.e., we simply write  $a$  instead of  $(a)$  for  $(a) \in (\text{Vars} \cup \Gamma)^1$ . Using this identification, a 1-dimensional trace over  $\Gamma$  precisely corresponds to the notion of *trace over  $\Gamma$*  used in [Kle+22]; and the syntax and semantics of 1-swg-queries precisely coincides with the syntax and semantics of the *swg-queries over  $\text{Vars}$  and  $\Gamma$*  introduced and studied in [Kle+22]. Hence, the notions introduced in Section 2.1 are a natural generalisation of the notions of [Kle+22] from dimension 1 to arbitrary dimension  $k \in \mathbb{N}_{\geq 1}$ . Furthermore, all results achieved in [Kle+22] for swg-queries over  $\text{Vars}$  and  $\Gamma$  immediately carry over to the 1-swg-queries over  $\text{Vars}$  and  $\Gamma$  considered in the current paper. A brief survey of [Kle+22] can be found in [Sch22].

The following example illustrates the correspondence between the swg-queries of [Kle+22] and 1-swg-queries.

**Example 4.** *The sequence of data items depicted in Figure 1a corresponds to the 1-trace*

$$t := (\text{SUB}) (\text{SCH}) (\text{EVI}) (\text{SCH}) (\text{KIL}) (\text{UPD}) (\text{CHE}) (\text{UPD}) (\text{SCH}) (\text{FIN})$$

which, by omitting brackets around 1-tuples, we shortly write as

$$\text{SUB SCH EVI SCH KIL UPD CHE UPD SCH FIN}$$

and this is a trace in the sense of [Kle+22]. The query depicted in Figure 1b searches for a subsequence of data items that indicates that a task was scheduled, killed, and, after treated in the same way twice, scheduled again; all within a global window size of at most 15 data items. This can be expressed as a 1-swg-query  $q = (s, w, c)$  as follows: The query string length is  $\ell := 5$ . The window size is  $w := 15$ . As there are no particular constraints on the gap sizes between the data items, the gap size constraints are chosen as  $c := ((0, \infty), (0, \infty), (0, \infty), (0, \infty))$  (meaning that each gap can be of arbitrary length). The query string is  $s := (\text{SCH}) (\text{KIL}) (x) (x) (\text{SCH})$ , which, by omitting brackets around 1-tuples, is identified with  $\text{SCH KIL } x x \text{ SCH}$ , and this exactly yields a swg-query as considered in [Kle+22]. We observe that  $t \models q$ , and a witness substitution  $\mu$  and embedding  $e$  can be illustrated as follows:

$$\begin{array}{cccccccccccc} s = & & \text{SCH} & & & \text{KIL} & x & & x & & \text{SCH} & & \\ t = & \text{SUB} & \text{SCH} & \text{EVI} & \text{SCH} & \text{KIL} & \text{UPD} & \text{CHE} & \text{UPD} & \text{SCH} & \text{FIN} & & \end{array}$$

### 2.3 A one-dimensional representation of multi-dimensional traces and queries

This subsection fixes an encoding that allows to represent  $k$ -dimensional traces and  $k$ -swg-queries over  $\text{Vars}$  and  $\Gamma$  by corresponding 1-dimensional traces and 1-swg-queries over  $\text{Vars}$  and a slightly extended type set  $\tilde{\Gamma}$ . This will allow us to transfer the results obtained in [Kle+22] for the 1-dimensional case to the multi-dimensional setting.

We let  $\tilde{\Gamma} := \Gamma \cup \{\#\}$  where  $\#$  is a new symbol that belongs neither to  $\Gamma$  nor to  $\text{Vars}$ . We will use  $\#$  as a separator to mark the beginning of the encoding of every  $k$ -dimensional data item. For each  $k$ -dimensional data item  $d = (a_1, \dots, a_k) \in \Gamma^k$  we let  $\text{enc}(d)$  be the 1-dimensional trace over  $\tilde{\Gamma}$  of length  $k+1$  defined as  $\text{enc}(d) := \# a_1 \cdots a_k$  (recall from Section 2.2 that we omit brackets around 1-dimensional data items, i.e.,  $\text{enc}(d)$  is  $(\#) (a_1) \cdots (a_k)$ ).

We lift  $\text{enc}$  to be a mapping from  $k$ -traces over  $\Gamma$  to 1-traces over  $\tilde{\Gamma}$  in the canonical way: for a  $k$ -trace  $t = t_1 t_2 \cdots t_n$  with  $t_i \in \Gamma^k$  for all  $i \in [n]$  we let  $\text{enc}(t) := \text{enc}(t_1) \text{enc}(t_2) \cdots \text{enc}(t_n)$ . Note that the 1-trace  $\text{enc}(t)$  has length  $(k+1) \cdot |t|$ .

The following example illustrates how an embedding  $e$  of a  $k$ -trace  $s$  in a  $k$ -trace  $t$  (witnessing that  $s \leq t$ ) can be transferred into an embedding  $\tilde{e}$  of the 1-trace  $\text{enc}(s)$  in the 1-trace  $\text{enc}(t)$ .

**Example 5.** Let  $k = 2$  and  $\Gamma = \{a, b, c\}$ . Consider the following 2-traces  $s$  and  $t$ :

$$\begin{array}{cccccc} s = & & (a, b) & & (a, b) & (b, c) & & (a, c) \\ t = & (a, b) & (a, b) & (a, c) & (a, b) & (b, c) & (a, b) & (a, c) \end{array}$$

Note that  $s \preceq_e t$ , witnessed by the embedding  $e$  illustrated above. I.e.,  $e : [4] \rightarrow [7]$  with  $e(1) = 2$ ,  $e(2) = 4$ ,  $e(3) = 5$ , and  $e(4) = 7$ . The 1-traces  $\tilde{s} := \text{enc}(s)$  and  $\tilde{t} := \text{enc}(t)$  are

$$\begin{array}{cccccccc} \tilde{s} = & & \# & a & b & & \# & a & b & \# & b & c & & \# & a & c \\ \tilde{t} = & \# & a & b & \# & a & b & \# & a & c & \# & a & b & \# & b & c & \# & a & b & \# & a & c \end{array}$$

Observe that  $\tilde{s} \preceq_{\tilde{e}} \tilde{t}$  by the embedding  $\tilde{e}$  illustrated above. This embedding is obtained from  $e$  by translating each position of a  $k$ -tuple in the  $k$ -trace into a block of  $k+1$  consecutive positions in the corresponding 1-trace.

Note that there also exist other embeddings of  $\tilde{s}$  in  $\tilde{t}$  that do not correspond to embeddings of  $s$  in  $t$ ; an example is the embedding  $\hat{e}$  illustrated as follows:

$$\begin{array}{cccccccc} \tilde{s} = & \# & a & & b & & \# & a & b & \# & b & c & & \# & a & c \\ \tilde{t} = & \# & a & b & \# & a & b & \# & a & c & \# & a & b & \# & b & c & \# & a & b & \# & a & c \end{array}$$

The following notion is a straightforward generalization of the way the embedding  $\tilde{e}$  was obtained from  $e$  in Example 5.

Let  $m, n \in \mathbb{N}_{\geq 1}$  and let  $e : [m] \rightarrow [n]$  such that  $e(i) < e(j)$  for all  $i, j \in [m]$  with  $i < j$ . Recall that  $k \in \mathbb{N}_{\geq 1}$  is the fixed *dimension*. We let  $\text{rep}_k(e)$  be the mapping from  $[(k+1)m]$  to  $[(k+1)n]$  defined as follows. We subdivide  $[(k+1)m]$  into  $m$  consecutive blocks of length  $(k+1)$  each — the  $i$ -th block starting at position  $(i-1)(k+1) + 1$  and ending at position  $i(k+1)$ , for every  $i \in [m]$ . The  $i$ -th block of  $[(k+1)m]$  is mapped by  $\text{rep}_k(e)$  onto the  $e(i)$ -th block of  $[(k+1)n]$ . I.e., for all  $i \in [m]$  and all  $p \in \{1, \dots, k+1\}$  we let

$$\text{rep}_k(e)((i-1)(k+1) + p) := (e(i)-1)(k+1) + p.$$

The following lemma provides the property intended by the choice of the definition of  $\text{rep}_k(e)$ ; the proof is straightforward and therefore omitted in this paper.

**Lemma 6.** Let  $s$  and  $t$  be two  $k$ -traces over  $\Gamma$  and let  $\tilde{s} := \text{enc}(s)$  and  $\tilde{t} := \text{enc}(t)$  be the corresponding 1-traces over  $\tilde{\Gamma}$ . If  $e$  is an embedding of  $s$  in  $t$  (witnessing that  $s \preceq_e t$ ), then  $\tilde{e} := \text{rep}_k(e)$  is an embedding of  $\tilde{s}$  in  $\tilde{t}$  (witnessing that  $\tilde{s} \preceq_{\tilde{e}} \tilde{t}$ ).

Next, we focus on how to translate a  $k$ -swg-query  $q = (s, w, c)$  (over  $\text{Vars}$  and  $\Gamma$ ) into a 1-swg-query  $\text{enc}(q) = (\tilde{s}, \tilde{w}, \tilde{c})$  over  $\text{Vars}$  and  $\tilde{\Gamma}$  in such a way that for all  $k$ -traces  $t$  we have:  $t \models q \iff \text{enc}(t) \models \text{enc}(q)$ .

The choices of  $\tilde{w}$  and  $\tilde{s}$  are obvious: We let

$$\tilde{w} := \text{rep}_k(w) := \begin{cases} \infty & \text{if } w = \infty \\ (k+1)w & \text{otherwise.} \end{cases}$$

The query string  $\tilde{s}$  is obtained from  $s$  in the analogous way as  $\text{enc}(t)$  is obtained from  $t$ . I.e., every  $k$ -tuple  $d = (a_1, \dots, a_k) \in (\text{Vars} \cup \Gamma)^k$  is mapped to  $\text{enc}(d) = \#a_1 \cdots a_k$ , and  $s = s_1 \cdots s_\ell$  with  $s_i \in (\text{Vars} \cup \Gamma)^k$  for all  $i \in [\ell]$  is mapped to  $\tilde{s} := \text{enc}(s) := \text{enc}(s_1) \cdots \text{enc}(s_\ell)$ .

Note that each position  $i$  of  $s$  now corresponds to  $k+1$  consecutive positions in  $\tilde{s}$ . The gap-size constraints in  $\tilde{c}$  are chosen in such a way that they ensure that the  $k$  gaps between these positions are of size exactly 0. Consequently, we let

$$\tilde{c} := \text{rep}_k(c) = \left( \underbrace{(0, 0), \dots, (0, 0)}_{k \text{ times } (0,0)}, \tilde{c}_1, \underbrace{(0, 0), \dots, (0, 0)}_{k \text{ times } (0,0)}, \tilde{c}_2, \dots, \tilde{c}_{\ell-1}, \underbrace{(0, 0), \dots, (0, 0)}_{k \text{ times } (0,0)} \right)$$

where for each  $i \in [\ell-1]$  the component  $\tilde{c}_i = (\tilde{c}_i^-, \tilde{c}_i^+)$  is obtained from the  $i$ -th component  $c_i = (c_i^-, c_i^+)$  of  $c$  by letting

$$\tilde{c}_i^- := (k+1)c_i^- \quad \text{and} \quad \tilde{c}_i^+ := \begin{cases} \infty & \text{if } c_i^+ = \infty \\ (k+1)c_i^+ & \text{otherwise.} \end{cases}$$

The following theorem states that the above definitions indeed have the intended functionality.

**Theorem 7.** *For every  $k$ -swg-query  $q$  over  $\text{Vars}$  and  $\Gamma$  and every  $k$ -trace  $t$  over  $\Gamma$  we have:  $t \models q \iff \text{enc}(t) \models \text{enc}(q)$ .*

*Proof.* We prove the direction “ $\implies$ ” (the proof of the opposite direction is analogous).

Let  $q = (s, w, c)$  be a  $k$ -swg-query over  $\text{Vars}$  and  $\Gamma$  and let  $t$  be a  $k$ -trace over  $\Gamma$ . Let  $\tilde{q} := \text{enc}(q) = (\tilde{s}, \tilde{w}, \tilde{c})$  and  $\tilde{t} := \text{enc}(t)$  be the corresponding 1-swg-query and 1-trace. Let  $\ell := |s|$  and  $n := |t|$ . Assume that  $t \models q$ . I.e., there exists a substitution  $\mu : (\text{Vars} \cup \Gamma) \rightarrow \Gamma$  and an embedding  $e : [\ell] \rightarrow [n]$  that satisfies  $w$  and  $c$ , such that  $\mu(s) \leq_e t$ . In other words:  $(\mu, e)$  is a witness for  $t \models q$ .

We let  $\tilde{e} := \text{rep}_k(e)$ . And we define  $\tilde{\mu} : (\text{Vars} \cup \tilde{\Gamma}) \rightarrow \tilde{\Gamma}$  with  $\tilde{\mu}(x) := \mu(x)$  for all  $x \in \text{Vars}$  and  $\tilde{\mu}(a) := a$  for all  $a \in \tilde{\Gamma} = \Gamma \cup \{\#\}$ . We claim that  $(\tilde{\mu}, \tilde{e})$  is a witness for  $\tilde{t} \models \tilde{q}$ . To prove this, we have to show that  $\tilde{\mu}(\tilde{s}) \leq_{\tilde{e}} \tilde{t}$  and that  $\tilde{e}$  satisfies  $\tilde{w}$  and  $\tilde{c}$ .

Let us start with the first task. By assumption we know that  $\mu(s) \leq_e t$ . From Lemma 6 we obtain that  $\text{enc}(\mu(s)) \leq_{\tilde{e}} \tilde{t}$ . Therefore, we are done by noting that  $\text{enc}(\mu(s)) = \tilde{\mu}(\tilde{s})$ .

Let  $\tilde{\ell} := |\tilde{s}|$ . Let us now verify that  $\tilde{e}$  satisfies  $\tilde{w}$ . In case that  $w = \infty$ , this is obvious. Let us focus on the case where  $w \neq \infty$ . By assumption we know that  $e$  satisfies  $w$ . We have:

$$\begin{aligned} \tilde{e}(\tilde{\ell}) - \tilde{e}(1) + 1 &= e(\ell)(k+1) - ((e(1)-1)(k+1) + 1) + 1 \\ &= (k+1) \cdot (e(\ell) - e(1) + 1) \\ &\leq (k+1) \cdot w = \tilde{w}. \end{aligned}$$

Finally, let us verify that  $\tilde{e}$  satisfies  $\tilde{c}$ . Note that  $\tilde{c}$  contains  $\ell-1+\ell\cdot k = (k+1)\cdot\ell-1 = \tilde{\ell}-1$  gap-size constraints, where  $\ell-1$  constraints correspond to the constraints  $c_i$  in  $c = (c_1, \dots, c_{\ell-1})$ , which got multiplied by  $(k+1)$ . The remaining  $\ell\cdot k$  constraints in  $\tilde{c}$  are equal to  $(0, 0)$ . Our definition of  $\tilde{e}$  ensures that the  $(0, 0)$ -constraints are satisfied.

By assumption we know that  $e$  satisfies  $c$ . Hence, for each  $i \in [\ell-1]$  we have:  $c_i^- \leq g_i \leq c_i^+$  for the actual size of the  $i$ -th gap  $g_i := e(i+1) - 1 - e(i)$ . Note that the size of the corresponding gap in the 1-dimensional representation is  $(k+1)g_i$ . This implies that the corresponding gap-size constraints in  $\tilde{c}$  are satisfied.

This completes the proof of the “ $\implies$ ”-direction of Theorem 7.  $\square$

Theorem 7 serves as a tool to lift results known for the 1-dimensional case to the multi-dimensional setting. In the next section, we implement this for the results on the query discovery problem obtained in [Kle+22].

### 3 Query Discovery and Implementation

The following notions were introduced in [Kle+22] for the 1-dimensional case and straightforwardly carry over to the multi-dimensional setting.

The *model set* of a  $k$ -swg-query  $q$  over  $\text{Vars}$  and  $\Gamma$  is  $\text{Mod}_\Gamma(q) := \{t \in (\Gamma^k)^+ : t \models q\}$ .

A query  $q'$  is said to be *strictly more restrictive than*  $q$  if  $\text{Mod}_\Gamma(q') \subsetneq \text{Mod}_\Gamma(q)$ .

A  $k$ -dimensional *sample* (over  $\Gamma$ ) is a finite, non-empty set  $\mathcal{S}$  of  $k$ -traces (over  $\Gamma$ ). The *support*  $\text{supp}(q, \mathcal{S})$  of a  $k$ -swg-query  $q$  in a sample  $\mathcal{S}$  is defined as the fraction of  $k$ -traces in  $\mathcal{S}$  that match  $q$ , i.e.,  $\text{supp}(q, \mathcal{S}) := \frac{|\{t \in \mathcal{S} : t \models q\}|}{|\mathcal{S}|}$ .

A *support threshold* is a rational number  $\text{sp}$  with  $0 < \text{sp} \leq 1$ . A query  $q$  is said to *cover* a sample  $\mathcal{S}$  with *support*  $\text{sp}$  if  $\text{supp}(q, \mathcal{S}) \geq \text{sp}$ .

Let us fix the query parameters  $(\ell, w, c)$  and a support threshold  $\text{sp}$ . Let  $\mathcal{S}$  be a sample. A  $k$ -swg-query  $q$  with parameters  $(\ell, w, c)$  is said to be *descriptive for*  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$  if  $q$  covers  $\mathcal{S}$  with support  $\text{sp}$ , and there is no  $k$ -swg-query  $q'$  with parameters  $(\ell, w, c)$  that is strictly more restrictive than  $q$  and that still covers  $\mathcal{S}$  with support  $\text{sp}$ . I.e.,  $\text{supp}(q, \mathcal{S}) \geq \text{sp}$  and there is no  $(\ell, w, c)$ -query  $q'$  such that  $\text{supp}(q', \mathcal{S}) \geq \text{sp}$  and  $\text{Mod}_\Gamma(q') \subsetneq \text{Mod}_\Gamma(q)$ .

The remainder of this section as well as the subsequent Section 4 are devoted to the following query discovery problem for arbitrary dimension  $k \in \mathbb{N}_{\geq 1}$ : The input consists of a support threshold  $\text{sp}$ , a  $k$ -dimensional sample  $\mathcal{S}$ , and query parameters  $(\ell, w, c)$ . The goal is to compute a  $k$ -swg-query  $q$  with parameters  $(\ell, w, c)$  that is descriptive for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$ . An algorithm solving this discovery problem for the 1-dimensional case (i.e., where  $k = 1$ ) was presented in [Kle+22]. In Section 3.1 we utilize Theorem 7 to lift this algorithm to arbitrary dimension  $k \geq 1$ ; Section 3.2 gives a brief description of our implementation of this algorithm.

### 3.1 An algorithm solving the query discovery problem for arbitrary dimension $k$

Let  $k \in \mathbb{N}_{\geq 1}$  be the given dimension and let  $(\ell, w, c)$  be the given query parameters. Note that the most general  $k$ -swg-query with parameters  $(\ell, w, c)$  is the query  $q_{mg} = (s_{mg}, w, c)$  whose query string  $s_{mg}$  is of the form  $((x_{1,1}, \dots, x_{1,k}) (x_{2,1}, \dots, x_{2,k}) \cdots (x_{\ell,1}, \dots, x_{\ell,k}))$  where the  $x_{i,j}$  for  $i \in [\ell]$  and  $j \in [k]$  are  $\ell \cdot k$  pairwise distinct variables in  $\text{Vars}$ . It is straightforward to see that  $\text{Mod}_{\Gamma}(q') \subseteq \text{Mod}_{\Gamma}(q_{mg})$  for every  $k$ -swg-query  $q'$  with parameters  $(\ell, w, c)$ .

The discovery algorithm takes as input a  $k$ -dimensional sample  $\mathcal{S}$ , a support threshold  $\text{sp}$ , and the query parameters  $(\ell, w, c)$ .

If  $\text{supp}(q_{mg}, \mathcal{S}) < \text{sp}$ , then there does not exist any  $k$ -swg-query  $q$  with parameters  $(\ell, w, c)$  with  $\text{supp}(q, \mathcal{S}) \geq \text{sp}$ , let alone a query that is descriptive for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$ . Therefore, the algorithm can safely abort with an error message indicating that the desired query does not exist.

If, on the other hand,  $\text{supp}(q_{mg}, \mathcal{S}) \geq \text{sp}$ , then the algorithm searches for an admissible *replacement operation* for each variable  $x \in \text{vars}(q_{mg})$ . Such an operation replaces  $x$  by a symbol  $y$  (which can be a type or an *available variable*). It is admissible if the resulting query  $q'$  satisfies  $\text{supp}(q', \mathcal{S}) \geq \text{sp}$ . If no replacement operation is possible, we keep  $x$ , i.e., the current query string remains unchanged, and  $x$  becomes available. After each variable  $x \in \text{vars}(q_{mg})$  has been considered, the algorithm terminates and produces the current query as output.

Pseudocode implementing this is provided in Algorithm 1. We start by letting  $q$  be the most general  $k$ -swg-query with parameters  $(\ell, w, c)$  and we let  $s$  be the query string of  $q$ . If  $q$  does not cover  $\mathcal{S}$  with support  $\text{sp}$ , we abort and return the message  $\perp$ , indicating that there does not exist any  $k$ -swg-query with parameters  $(\ell, w, c)$  that is descriptive for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$ . Otherwise, we proceed by letting  $\Delta$  be the set of types that satisfy the support threshold (these will be the types available for replacement operations), and we initialise the set  $U$  of *unvisited* variables to be the set of all variables occurring in  $q$ . The set  $V$  of *available* variables is initialized to be the empty set. During the main loop in line 5, each variable  $x \in U$  is considered exactly once, and for each such variable, the algorithm tests whether there exists a type or variable  $y$  from  $\Omega := (\Delta \cup V)$  such that replacing all occurrences of variable  $x$  by  $y$  is an admissible replacement operation.<sup>7</sup> If such an  $y \in \Omega$  exists, we perform the actual replacement in line 12. Otherwise, no replacement operation is possible, hence we do not change the current query string but add  $x$  to the set  $V$  of available variables (line 16).

For dimension  $k = 1$ , this algorithm was presented in [Kle+22].

Note that the lines 6 and 9 of Algorithm 1 allow to make an arbitrary choice. Different choices lead to different “runs” of the algorithm, and different runs might produce different output queries.

---

<sup>7</sup> We write  $s(x \mapsto y)$  to denote the query string obtained from  $s$  by replacing every occurrence of the variable  $x$  by the symbol  $y$ .

---

**ALGORITHM 1:** ComputeDescriptiveQuery( $\mathcal{S}, \text{sp}, (\ell, w, c)$ )

---

**Input** :  $k$ -dim. sample  $\mathcal{S}$ ; support threshold  $\text{sp}$  with  $0 < \text{sp} \leq 1$ ; query parameters  $(\ell, w, c)$   
**Returns** : descriptive  $k$ -swg-query  $q$  for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$  or error message  $\perp$

```

1  $s := s_{mg}; q := (s, w, c)$  // query string and query; start with the most general query
2 if  $\text{supp}(q, \mathcal{S}) < \text{sp}$  then stop and return  $\perp$ 
3  $\Delta := \{\gamma \in \Gamma : \frac{|\{t \in \mathcal{S} : \gamma \in \text{types}(t)\}|}{|\mathcal{S}|} \geq \text{sp}\}$  // types to be considered
4  $U := \text{vars}(q); V := \emptyset$  // unvisited variables and available variables
5 while  $U \neq \emptyset$  do
6     select an arbitrary  $x \in U$  and let  $U := U \setminus \{x\}$ 
7      $\Omega := (\Delta \cup V)$ ;  $\text{replace} := \text{False}$  // available symbols
8     while  $\Omega \neq \emptyset$  do
9         select an arbitrary  $y \in \Omega$  and let  $\Omega := \Omega \setminus \{y\}$ 
10         $q' := (s\langle x \mapsto y \rangle, w, c)$ 
11        if  $\text{supp}(q', \mathcal{S}) \geq \text{sp}$  then
12             $s := s\langle x \mapsto y \rangle$  // ReplaceOp
13             $\text{replace} := \text{True}$ 
14            break inner loop
15        if  $\text{replace}$  is False then
16             $V := V \cup \{x\}$  // NoChangeOp
17 stop and return  $q := (s, w, c)$ 

```

---

**Theorem 8.** Let  $k \in \mathbb{N}_{\geq 1}$ , let  $\mathcal{S}$  be a  $k$ -dimensional sample, let  $\text{sp}$  be a support threshold with  $0 < \text{sp} \leq 1$ , and let  $(\ell, w, c)$  be query parameters.

- (a) If there does not exist any  $k$ -swg-query with parameters  $(\ell, w, c)$  that is descriptive for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$ , then there is only one run of Algorithm 1 upon input  $\mathcal{S}, \text{sp}, (\ell, w, c)$ , and this run stops in line 2 with output  $\perp$ .
- (b) Otherwise, every run of Algorithm 1 upon input  $\mathcal{S}, \text{sp}, (\ell, w, c)$  terminates and outputs a query that is descriptive for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$ .

*Proof sketch.* For the special case where  $k = 1$ , the theorem was proved in [Kle+22]. Moreover, [Kle+22] provided the following slightly stronger result — again, for the 1-dimensional case: The algorithm obtained from Algorithm 1 by omitting line 1 and, instead, starting with an arbitrary input query  $q$ , outputs either a query  $q'$  that is descriptive for  $\mathcal{S}$  w.r.t.  $(\text{sp}, (\ell, w, c))$  and satisfies  $\text{Mod}_\Gamma(q') \subseteq \text{Mod}_\Gamma(q)$  or, in case that no such  $q'$  exists, the message  $\perp$ . We use this for the particular 1-dimensional input query  $q := \text{enc}(q_{mg})$ , where  $q_{mg}$  is the most general  $k$ -swg-query with parameters  $(\ell, w, c)$  for an arbitrary dimension  $k \geq 2$ . Utilizing the one-dimensional representation of  $k$ -dimensional traces and queries presented in Section 2.3 and, in particular, Theorem 7, this yields the theorem's statement for arbitrary dimension  $k \geq 2$ .  $\square$

The above theorem guarantees that the output produced by Algorithm 1 is correct in the sense that it is either a query that is descriptive for its input or the message  $\perp$  indicating that no such query exists. Different runs of the algorithm may produce different queries (each with the guarantee that the delivered query is descriptive for its input). Let us mention, however, that (as shown in [Kle+22]) there exist inputs for which there exist some queries that cannot be delivered by any run of Algorithm 1 but that are descriptive for the input.

We close this subsection with two example runs of Algorithm 1.

**Example 9.** *For simplicity, the example deals with dimension  $k = 1$ . Let  $\Gamma = \{a, b, c\}$ ,  $\mathcal{S} = \{a b b, a c c\}$ ,  $\text{sp} = 1$ ,  $\ell = w = 3$ , and  $c = ((0, 0), (0, 0))$ . Upon this input, each run of Algorithm 1 lets  $s := s_{\text{mg}} = x_1 x_2 x_3$ , where  $x_1, x_2, x_3$  are three pairwise distinct variables in  $\text{Vars}$ . Since  $\text{supp}(q, \mathcal{S}) = 1$ , the algorithm proceeds by computing  $\Delta = \{a\}$  and letting  $U = \{x_1, x_2, x_3\}$  and  $V = \emptyset$ .*

*Let us assume that in the first transition through the outer loop the algorithm selects  $x := x_3$ . Due to  $\Omega = \{a\}$ , the only possible replacement is  $s \langle x_3 \mapsto a \rangle$  — but it turns out that this replacement is not admissible as its support on  $\mathcal{S}$  is  $< 1$ . Hence,  $x_3$  remains unchanged in the query string and is inserted in the set  $V$  of available variables, i.e.,  $V = \{x_3\}$ .*

*Let us assume that in the second transition through the outer loop the algorithm selects  $x := x_1$ , and in the inner loop it selects  $y := a$ . It turns out that the replacement of  $x_1$  by  $a$  is admissible (as it has support 1 on  $\mathcal{S}$ ). Hence,  $s$  is replaced by the new query string  $a x_2 x_3$  and  $V$  remains unchanged.*

*In its last iteration through the outer loop, it turns out that  $s \langle x_2 \mapsto x_3 \rangle$  is the only admissible replacement operation. The algorithm’s run terminates after this iteration and outputs the query  $q = (s, w, c)$  with  $s = a x_3 x_3$ .*

*We illustrate this entire run as follows:*

$$x_1 \ x_2 \ \underline{x_3} \quad \overset{\Delta=\{a\}, V=\emptyset}{\rightsquigarrow} \quad \underline{x_1} \ x_2 \ x_3 \quad \overset{\Delta=\{a\}, V=\{x_3\}}{\rightsquigarrow} \quad a \ \underline{x_2} \ x_3 \quad \overset{\Delta=\{a\}, V=\{x_3\}}{\rightsquigarrow} \quad a \ x_3 \ x_3$$

*Another run (that outputs a very similar query) is:*

$$x_1 \ \underline{x_2} \ x_3 \quad \overset{\Delta=\{a\}, V=\emptyset}{\rightsquigarrow} \quad \underline{x_1} \ x_2 \ x_3 \quad \overset{\Delta=\{a\}, V=\{x_2\}}{\rightsquigarrow} \quad a \ \underline{x_2} \ x_3 \quad \overset{\Delta=\{a\}, V=\{x_2\}}{\rightsquigarrow} \quad a \ x_2 \ x_2$$

### 3.2 Implementation

In this section, we describe a prototypical implementation of our approach as a stand-alone Python tool, which is publicly available.<sup>8</sup> We explain how we express multi-dimensional samples and queries. We also briefly discuss how to decide whether  $\text{supp}(q, \mathcal{S}) \geq \text{sp}$  for given  $\mathcal{S}$ ,  $\text{sp}$  and  $q$  in Algorithm 1. Our implementation is designed for 1-dimensional as well as multi-dimensional queries and samples; in the context of this paper we focus on the multi-dimensional setting.

<sup>8</sup> <https://gitlab.com/kleemeis95/sfb-1404-fonda-querydiscovery-prototype>

**Samples.** We model traces as Python strings and use dedicated characters to separate data items as well as data item attributes. Moreover, a (multi-dimensional) sample  $\mathcal{S}$  is described by an instance of a specific class (`MultidimSample`). It combines a list of traces (the actual sample) with meta information, e.g., the sample size (i.e., the number of traces), the data item dimension, and the set of all types occurring in  $\mathcal{S}$  (optionally partitioned by the data item attributes and filtered by a given support threshold).

**Queries.** A query  $q = (s, w, c)$  is implemented as an instance of a respective class (`MultidimQuery`). It encapsulates the query string (a Python string, using different separator symbols), the global window size (an integer), the local gap-size constraints (a list of tuples), and meta information, e.g. the set of types which occur within  $s$ , and the set of variables that occur more than once in  $s$ .

**Matching.** The matching routine is implemented as part of the class `MultidimQuery`. Here, a function `match_sample` takes as input a `MultidimSample`-instance  $\mathcal{S}$  and a support threshold `sp`, and outputs true, if  $\text{supp}(q, \mathcal{S}) \geq \text{sp}$ , whereby  $q$  denotes the query which is represented by the current class instance. During a run of `match_sample`, we start by transforming the query string into a regular expression according to the Python `re` module, which takes into account the gap-size constraints  $c$ . We then test whether  $t \models q$ , for each  $t \in \mathcal{S}$ , by using the function `search` provided by Python for regular expressions.

**Discovery Algorithm.** Algorithm 1 is implemented according to the pseudocode presented in Section 3.1. However, we allow the user to influence the arbitrary choices in lines 6 and 9 of the algorithm: one may choose the next, not yet visited variable in the query string either (i) arbitrarily (as described in Section 3.1) or (ii) by scanning the query string from left to right, or vice versa. Furthermore, types (available variables) may be preferred over available variables (types), instead of making an arbitrary choice.

## 4 Experimental Evaluation

Using the prototype introduced above, we applied our approach to real-world data in the domain of cluster monitoring, a scenario that we already used as a running example. Our goal has been to assess the general feasibility of our approach to the discovery of descriptive queries for multi-dimensional sequence data. We summarize our results as follows:

- We have been able to discover queries from the real-world data, thereby providing evidence that query discovery is indeed feasible. We note though, that the number of discovered queries that cannot immediately be linked to situations of interest is very large, since the given traces consist of many regularities that materialize in the discovered queries.
- Comparing the runtime of our approach for different data samples, it turned out that, in addition to the dataset size, the traversal of the query string and the need to assess variable operations have a major impact.

Below, we first describe our experimental setup in Section 4.1, before we discuss the obtained results in Section 4.2.

## 4.1 Experimental Setup

**Datasets.** Our experiments used the Google Cluster Traces [RWH11], a dataset that contains cell information over multiple days. Cells are sets of machines that share a cluster-management system. The machines handle incoming jobs, which consist of at least one task. The dataset contains six types of tables, capturing information about machines, jobs, tasks, constraints, and resources. For a detailed description of the dataset, we refer the reader to [RWH11].

For our experiments, we considered the information on task executions. Specifically, each data item includes the following attributes:

- (1) **job:** The job to which the task belongs. Each job is assigned a unique 64-bit identifier.
- (2) **task:** The task index within a job, given as an integer value.
- (3) **machine:** The machine on which the task shall run. Each machine is assigned a unique 64-bit identifier.
- (4) **status:** The task's status in terms of its lifecycle. It is encoded an integer  $i \in \{0, \dots, 8\}$  that corresponds to one of the following states: SUBMIT (0), SCHEDULE (1), EVICT (2), FAIL (3), FINISH (4), KILL (5), LOST (6), UPDATE\_PENDING (7), or UPDATE\_RUNNING (8).
- (5) **priority:** The execution priority of the task, modelled as an integer. The larger the number, the higher the priority.

To achieve a controlled setup for query discovery, we employed a setup that is based on three pre-defined queries  $q_i$  for  $i \in [3]$ . The idea being that based on the matches of such a query  $q_i$ , we derive a sample of traces  $\mathcal{S}_i$ , so that our discovery algorithm can be expected to discover query  $q'_i$ , with  $\text{Mod}(q'_i) \subseteq \text{Mod}(q_i)$ , when using a support value of  $\text{sp} = 1.0$ . We realised this approach with the following three queries:

```
q1: PATTERN SEQ(Task a, Task b, Task c)
    WHERE a.status = c.status = 1 AND b.status = 5
    AND a.job = b.job = c.job AND a.machine = b.machine = c.machine
    WITHIN 1000 data items
q2: PATTERN SEQ(Task a, Task b, Task c, Task d)
    WHERE a.status = b.status = c.status = d.status = 4
    AND a.machine = b.machine = c.machine = d.machine
    WITHIN 100 data items
q3: PATTERN SEQ(Task a, Task b, Task c)
    WHERE a.machine = b.machine AND a.job=c.job
    AND a.status = b.status = 1 AND c.status=2
    WITHIN 100 data items
```

Based on the matches of these queries, we constructed the samples  $\mathcal{S}_i$ ,  $i \in [3]$ , as follows: For samples  $\mathcal{S}_1$  and  $\mathcal{S}_2$ , the end of a trace was determined by the last item of a match. The

start of a trace was defined as the item following the last item of the *previous* match (or the first item of the whole dataset, respectively), so that traces are non-overlapping, i.e., any data item appears in at most one of the traces of a sample. Moreover, the traces for samples  $\mathcal{S}_1$  and  $\mathcal{S}_2$  were partitioned by the `machine` attribute, as the respective queries refer solely to items within such a partition. Finally, all traces with at most 100 items were included in the sample, which selected more than 98% ( $\mathcal{S}_1$ ) or 91% ( $\mathcal{S}_2$ ) of the constructed traces. For sample  $\mathcal{S}_3$ , again, the end of a trace was determined by the last item of a match. The start of a trace, however, was defined to be the first item of a match. The constructed samples had the following characteristics:

| Sample          | Size | Min. trace length | Max. trace length |
|-----------------|------|-------------------|-------------------|
| $\mathcal{S}_1$ | 558  | 3                 | 96                |
| $\mathcal{S}_2$ | 679  | 22                | 99                |
| $\mathcal{S}_3$ | 84   | 4                 | 197               |

**Experimental Procedure and Measures.** For each sample, Algorithm 1 was executed using the most general query, setting its parameters ( $\ell, w, c$ ) as follows. For the length of the query string, we set  $\ell = 3$  for  $\mathcal{S}_1$  and  $\mathcal{S}_3$ , and  $\ell = 4$  for  $\mathcal{S}_2$ . The global window size was set to  $w = 100$  for  $\mathcal{S}_1$  and  $\mathcal{S}_2$ , and to  $w = 200$  for  $\mathcal{S}_3$ , as derived from the sample generation procedure. We defined the local gap-size constraints to be least restrictive by setting them to  $c = ((0, 100), (0, 100))$  for  $\ell = 3$ , and to  $c = ((0, 200), (0, 200), (0, 200))$  for  $\ell = 4$ , respectively. We considered several support thresholds (namely, 0.6, 0.8, and 1.0).

Concerning the choices in Algorithm 1 on the next unvisited variable and replacement preference, we consider all options outlined in Section 3.2: the next variable is derived left-to-right (l2r), right-to-left (r2l), or arbitrarily (a). This choice is combined with no preference (a), or preference given to types (t) or variables (v).

In addition to illustrating some of the discovered queries, we measure the runtime of our approach for different instantiations. We further break down the runtime by profiling the our implementation with Python’s cProfile to shed light on the contribution of the various algorithmic steps.

**Experimental Environment.** All experiments have been executed on a 64 Bit Manjaro system with an AMD Ryzen 5 Pro 5650U processor running at 4.1 GHz and 16GB RAM.

## 4.2 Evaluation

**Discovered queries.** Applying our approach for each sample  $\mathcal{S}_i, i \in [3]$ , we successfully discovered multi-dimensional swg-queries. Moreover, the discovered queries turned out to

be more specific than the queries used to create the samples (see Section 4.1). For instance, we discovered the following query strings for  $sp = 1.0$  for the three samples:

$$\begin{aligned} \mathcal{S}_1 : \quad s_1 &= (x_j, x_t, x_m, 1, x_p)(x_j, x_t, x_m, 5, x_p)(x_j, y_1, x_m, 1, y_2) \\ \mathcal{S}_2 : \quad s_2 &= (y_1, y_2, x_m, 4, x_p)(y_3, y_4, x_m, 4, x_p)(y_5, y_6, x_m, 4, x_p)(y_7, y_8, x_m, 4, x_p) \\ \mathcal{S}_3 : \quad s_3 &= (x_j, y_1, x_m, 1, x_p)(y_2, y_3, x_m, 1, y_4)(x_j, y_5, y_6, 2, x_p) \end{aligned}$$

whereby  $x_j, x_t, x_m, x_p, y_1, \dots, y_8$  are pairwise distinct variables in Vars. We note that these query strings can be used to derive queries in common languages for complex event recognition. Taking  $s_1$  as an example, we derive the following query  $q'_1$ :

```
q1':  PATTERN SEQ(Task a, Task b, Task c)
      WHERE a.status = c.status = 1 AND b.status = 5
      AND   a.job = b.job = c.job AND a.machine = b.machine = c.machine
      AND   a.task = b.task AND a.priority = b.priority
      WITHIN 1000 data items
```

Comparing query  $q'_1$  with query  $q_1$  from Section 4.1, we observe that the last line of the WHERE-clause renders the discovered query more specific. Similar observations are done for the descriptive queries discovered for the other samples. For instance, the above query strings  $s_2$  and  $s_3$  enforce conditions on the `priority` attribute that have not been part of the queries used to generate the samples.

Our results indicate that it is feasible to discover multi-dimensional swg-queries from real-world data. However, we also note that we discovered descriptive queries that would not match the last item of a trace, i.e., the supposed situation of interest. This highlights that the discovered queries will still have to be assessed by domain experts.

**Runtime.** Figure 3-5 provide the results for our runtime measurements in seconds, when varying the support threshold and the configuration of Algorithm 1 for selecting the next unvisited variable and the replacement preference. Overall, a higher support threshold yields smaller runtimes. This is due to a smaller number of supported types that have to be tested as well as the fact that the match test stops as soon as the support threshold cannot be satisfied any more. Moreover, the fastest runs have in common that the algorithm goes through the query string from left to right (l2r), while there is no clear trend for the replacement preference. However, this result highlights the potential for improvements based on heuristics to guide the exploration of the search space.

**Performance profiling.** We illustrate the results of the performance profile for  $\mathcal{S}_3$  in Figure 6, using a SnakeViz' icicle plot [sna]. Here, each rectangle represents a function, while the layering of rectangles captures the call hierarchy between functions (the top function is called first). Moreover, each function is annotated with its overall runtime, which is visualized by the width of the rectangle.

We observe that the runtime of the main function of our experiments is dominated by the function `compute_descr_swgquery`, which implements Algorithm 1. It has to decide whether  $\text{supp}(q, \mathcal{S}) \geq sp$  multiple times, i.e., for each unvisited variable  $x$  the algorithm

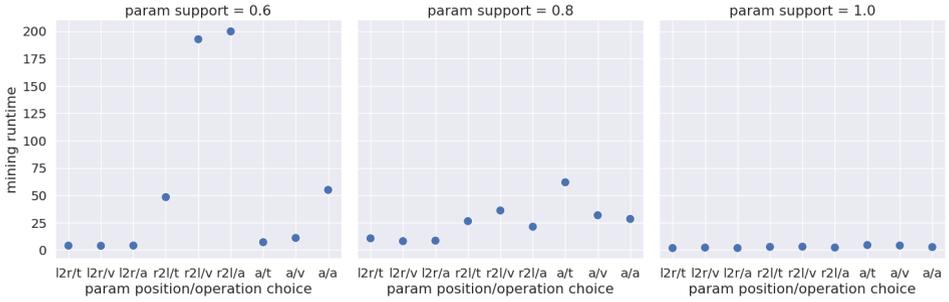


Fig. 3: Runtime measurements for sample  $S_1$ .

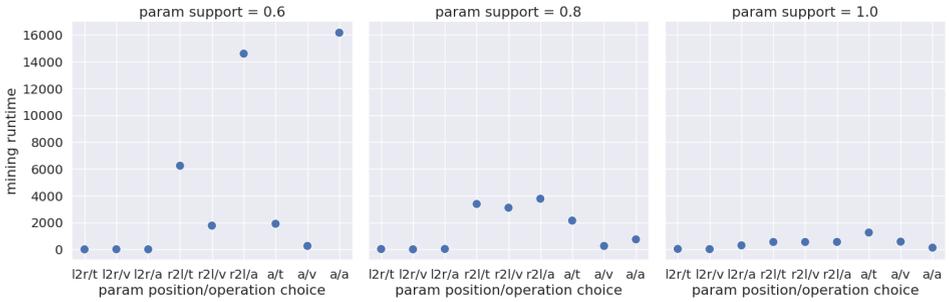


Fig. 4: Runtime measurements for sample  $S_2$ .

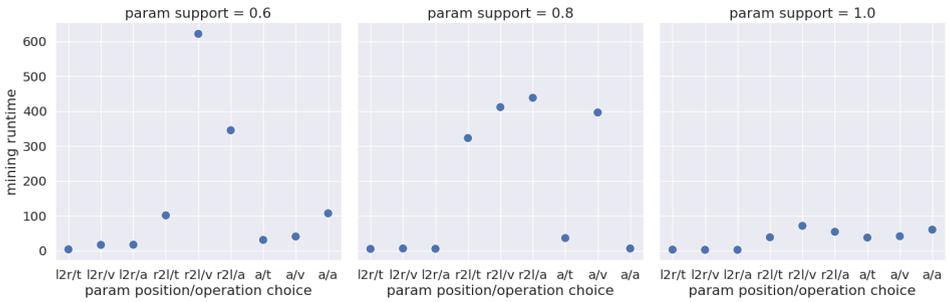


Fig. 5: Runtime measurements for sample  $S_3$ .

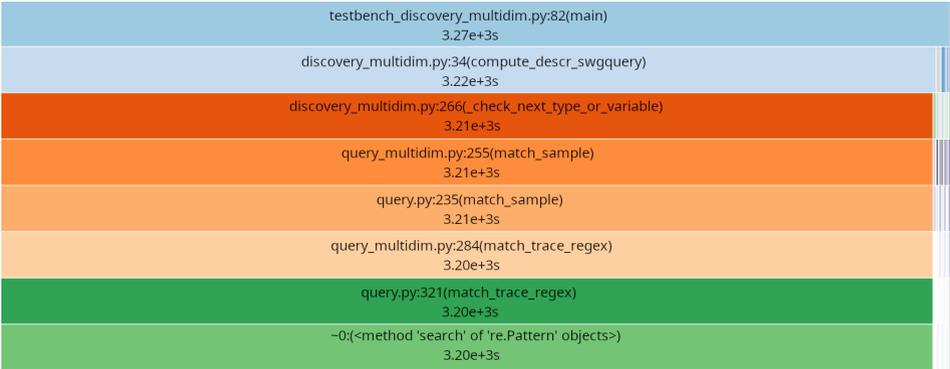


Fig. 6: Icicle visualisation of sample  $\mathcal{S}_3$ 's profile.

tests, whether there exists a type or variable so that the replacement yields a query satisfying the support threshold. This check is realised by function `_check_next_type_or_variable`. After building the regular expression corresponding to the current query, the matching problem is solved. Figure 6 illustrates that this function search of `re.Pattern` dominates the overall runtime. We conclude that performance improvements for query discovery may be achieved through optimizations of the function to decide whether a trace matches a query. In future work, we strive for algorithmic optimizations that exploit the fact that many subsequent match tests are conducted over the same set of traces with only slightly changed queries.

## 5 Concluding Remarks

Motivated by sequence data over a multi-dimensional schema, we defined an encoding to lift swg-queries and corresponding concepts as described in [Kle+22] to a multi-dimensional setting (Section 2). Furthermore we described our prototypical implementation of query discovery for multi-dimensional data (Section 3) and discussed experiments and their results on a real-world dataset (Section 4).

Our experiments' main result can be summarised as general feasibility of discovering swg-queries from multi-dimensional data: for each data set and configuration of Algorithm 1 we discovered a bunch of descriptive queries. This query set includes queries which are more specific than the queries we used for the data set generation.

Furthermore, our experiments suggest that structural characteristics of the sequence data plays an important role regarding the runtime of our discovery algorithm at various levels. Firstly, we observed notable differences in runtime regarding the way we select the next unvisited variable and its possible replacement. Hence we are interested in finding heuristics to predict which combination is most promising. Besides, we might be able to exploit the facts that  $q'$  does not change while testing  $\text{supp}(q', \mathcal{S}) \geq \text{sp}$  once and that  $\mathcal{S}$  does not change during the entire run of the discovery algorithm.

## References

- [Ang80] Dana Angluin. “Inductive Inference of Formal Languages from Positive Data”. In: *Inf. Control.* 45.2 (1980), pp. 117–135. DOI: 10.1016/S0019-9958(80)90285-5.
- [Art+14] Alexander Artikis et al. “Heterogeneous Stream Processing and Crowdsourcing for Urban Traffic Management”. In: *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014*. OpenProceedings.org, 2014, pp. 712–723. DOI: 10.5441/002/edbt.2014.77.
- [Bab+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. “Models and Issues in Data Stream Systems”. In: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*. ACM, 2002, pp. 1–16. DOI: 10.1145/543613.543615.
- [CM12] Gianpaolo Cugola and Alessandro Margara. “Processing flows of information: From data stream to complex event processing”. In: *ACM Comput. Surv.* 44.3 (2012), 15:1–15:62. DOI: 10.1145/2187671.2187677.
- [Day+21] Joel D. Day, Pamela Fleischmann, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. “The Edit Distance to k-Subsequence Universality”. In: *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*. 2021, 25:1–25:19. DOI: 10.4230/LIPIcs.STACS.2021.25.
- [Day+22] Joel D. Day, Maria Kosche, Florin Manea, and Markus L. Schmid. “Subsequences With Gap Constraints: Complexity Bounds for Matching and Analysis Problems”. In: vol. abs/2206.13896. 2022. DOI: 10.48550/arXiv.2206.13896. arXiv: 2206.13896.
- [Fer+18] Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. “Revisiting Shinohara’s algorithm for computing descriptive patterns”. In: *Theor. Comput. Sci.* 733 (2018), pp. 44–54. DOI: 10.1016/j.tcs.2018.04.035.
- [FR10] Dominik D. Freydenberger and Daniel Reidenbach. “Existence and nonexistence of descriptive patterns”. In: *Theor. Comput. Sci.* 411.34-36 (2010), pp. 3274–3286. DOI: 10.1016/j.tcs.2010.05.033.
- [FR13] Dominik D. Freydenberger and Daniel Reidenbach. “Inferring descriptive generalisations of formal languages”. In: *J. Comput. Syst. Sci.* 79.5 (2013), pp. 622–639. DOI: 10.1016/j.jcss.2012.10.001.
- [Gaw+21] Pawel Gawrychowski, Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. “Efficiently Testing Simon’s Congruence”. In: *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*. 2021, 34:1–34:18. DOI: 10.4230/LIPIcs.STACS.2021.34.

- [GCW16] Lars George, Bruno Cadonna, and Matthias Weidlich. “IL-Miner: Instance-Level Discovery of Complex Event Patterns”. In: *Proc. VLDB Endow.* 10.1 (2016), pp. 25–36. DOI: 10.14778/3015270.3015273.
- [Gia+20] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. “Complex event recognition in the Big Data era: a survey”. In: *VLDB J.* 29.1 (2020), pp. 313–352. DOI: 10.1007/s00778-019-00557-w.
- [Kle+22] Sarah Kleest-Meißner, Rebecca Sattler, Markus L. Schmid, Nicole Schweikardt, and Matthias Weidlich. “Discovering Event Queries from Traces: Laying Foundations for Subsequence-Queries with Wildcards and Gap-Size Constraints”. In: *25th International Conference on Database Theory, ICDT 2022*. Vol. 220. LIPIcs. 2022, 18:1–18:21. DOI: 10.4230/LIPIcs.ICDT.2022.18.
- [Kos+22a] Maria Kosche, Tore Koß, Florin Manea, and Viktoriya Pak. “Subsequences in Bounded Ranges: Matching and Analysis Problems”. In: *CoRR* abs/2207.09201 (2022). DOI: 10.48550/arXiv.2207.09201. arXiv: 2207.09201.
- [Kos+22b] Maria Kosche, Tore Koß, Florin Manea, and Stefan Siemer. “Combinatorial Algorithms for Subsequence Matching: A Survey”. In: *CoRR* abs/2208.14722 (2022). DOI: 10.48550/arXiv.2208.14722. arXiv: 2208.14722.
- [MCT14] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. “Learning from the past: automated rule generation for complex event processing”. In: *The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014*. ACM, 2014, pp. 47–58. DOI: 10.1145/2611286.2611289.
- [MS19] Florin Manea and Markus L. Schmid. “Matching Patterns with Variables”. In: *Combinatorics on Words - 12th International Conference, WORDS 2019, Loughborough, UK, September 9-13, 2019, Proceedings*. 2019, pp. 1–27. DOI: 10.1007/978-3-030-28796-2\_1.
- [RS97] Grzegorz Rozenberg and Arto Salomaa. “Patterns”. In: *Handbook of Formal Languages*. Vol. 1. Springer, 1997, pp. 230–242.
- [RWH11] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. “Google cluster-usage traces: format+ schema”. In: *Google Inc., White Paper* (2011), pp. 1–14.
- [SA95] Takeshi Shinohara and Setsuo Arikawa. “Pattern Inference”. In: *Algorithmic Learning for Knowledge-Based Systems, GOSLER Final Report*. 1995, pp. 259–291. DOI: 10.1007/3-540-60217-8\_13.
- [Sch22] Markus L. Schmid. “Extending Shinohara’s Algorithm for Computing Descriptive (Angluin-Style) Patterns to Subsequence Patterns”. In: *CoRR* abs/2206.13918 (2022). DOI: 10.48550/arXiv.2206.13918. arXiv: 2206.13918.

- [Shi82] Takeshi Shinohara. “Polynomial Time Inference of Pattern Languages and Its Application”. In: *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, MFCS*. 1982, pp. 191–209.
- [sna] snakeviz. *SnakeViz documentation*. URL: <https://jiffyclub.github.io/snakeviz/>.
- [TRP12] Kia Teymourian, Malte Rohde, and Adrian Paschke. “Knowledge-based processing of complex stock market events”. In: *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*. ACM, 2012, pp. 594–597. DOI: 10.1145/2247596.2247674.
- [Ver+15] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. “Large-scale cluster management at Google with Borg”. In: *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*. ACM, 2015, 18:1–18:17. DOI: 10.1145/2741948.2741964.



# Learn What Really Matters: A Learning-to-Rank Approach for ML-based Query Optimization

Henriette Behr<sup>1</sup> Volker Markl<sup>2</sup> Zoi Kaoudi<sup>3</sup>

**Abstract:** Query optimization is crucial for any data management system to achieve good performance. Recent advancements in Machine Learning (ML) have led to several efforts in the database research community that aim at improving query optimization with the help of ML. In particular, many works propose replacing the cost model used during plan enumeration with an ML model. The goal of these works is to learn a regression model from previously executed query plans that estimates the runtime of a given plan. Interestingly, it is well-known that what really matters in query optimization is the relative order of the query plan alternatives and not their actual cost or runtime. We thus take a learning-to-rank approach and propose a novel neural network model architecture that can predict the rank of a plan. It considers a plan in comparison with alternative plans of the same query and together with a loss function that incorporates ranking metrics into the learning process we highlight the learning-to-rank objective. To enable training, we first extract features from query plans by adapting a state-of-the-art deep learning approach so that all features are independent of the input dataset schema. Second, we devise two score functions that map the runtime of plans to scores which are then used as labels during training. We integrate the trained model into an adapted bottom-up plan enumeration algorithm that finds the best possible execution plan for a given query. We evaluate our approach against two state-of-the-art ML models and the highly tuned cost model of a commercial database and measure the runtime of the plans chosen in each case when executed in the database. We show that our approach achieves up to an order of magnitude better query performance than the comparison models and is able to either match (for short and medium-running queries) or outperform the commercial database (up to 5× for long-running queries).

**Keywords:** query optimization; learning-to-rank; cost model

## 1 Introduction

Recent advancements in machine/deep learning (ML)<sup>§</sup> have shed light to many open data management problems, ranging from indexing and query optimization to database tuning. As query optimization is at the core of any database system, there are several efforts towards using ML in various (or all) steps of the process, e.g., for cardinality estimation [Ya19, Ne21, Ki19], join ordering [Yu20, MP18], and cost modeling [Ma19, Ma21, MP19]. In particular for cost modeling, the main idea is to completely replace the cost model and statistics used in a query optimizer with an ML model. Following the term cost-based query optimization, we

---

<sup>1</sup> TU Berlin, Germany, henriette.behr.1@gmail.com. Work done while conducting master thesis in TU Berlin.

<sup>2</sup> TU Berlin, Germany, volker.markl@tu-berlin.de

<sup>3</sup> IT University of Copenhagen, Copenhagen, Denmark, zoka@itu.dk. Work done while at TU Berlin.

<sup>§</sup>We use the term ML to also refer to deep learning.

call such an optimization *learning-based or ML-based query optimization*. The benefit of ML-based query optimization compared to cost-based is twofold. First, the optimizer can learn more complex models that are not necessarily linear functions and can thus better depict the database's performance. Second, it mitigates the tedious and time-consuming task of manually tuning the cost model.

Recent works that follow the learning-based paradigm, such as [Ma19, Ma21, Ka20], build a regression model based on previously executed query plans (training data). Such models provide an estimation of a plan's runtime which is then used for plan enumeration and pruning. Even though current approaches focus on estimating the real cost of query plans, in the end *what really matters* for the query optimizer is their relative order: Is plan *A* faster than plan *B*? By considering just the order of the plans the optimizer can still prune inefficient ones, keep more promising ones, and select one that performs well. Based on this observation, we argue that it suffices to learn the order (rank) of query plans instead of estimating their runtimes. We, thus, devise an ML-based learning-to-rank (LTR) approach that ranks execution plans without estimating their execution time. Our approach scores execution plans of a given query and ranks them based on their scores. Although researchers have extensively used LTR models for information retrieval or recommendation [Li11], to our knowledge, an LTR approach has not been used for query optimization yet.

To reach this goal, we have to overcome some challenges. First, there is a wide spectrum of LTR approaches (pointwise, pairwise or listwise), loss functions, and neural network architectures that one has to choose from. We, thus, need to make design decisions in all these directions so that our solution performs at least as good as a highly-tuned cost-based optimizer, if not better. Second, although one could use the estimated runtimes to get a ranking over the plans, these are not only hard to estimate but also do not always preserve the ordering of the plans. To train an ML model that can rank execution plans, we need to devise a score function that uniformly ranks the different execution plans. Third, we need to study whether and how existing enumeration algorithms can be used with LTR models.

We tackle these challenges and make the following main contributions:

- (1) We devise a listwise LTR approach that is more suitable for ML-based query optimization. We consider several loss functions and a neural network architecture tailored to the query optimization problem (Section 3.1).
- (2) We propose two score functions, one local, which assigns scores to plans per query, and one global, which assigns scores across the plans of all training queries (Section 3.2).
- (3) We describe our featurization scheme and adapt a well-known bottom up plan enumeration algorithm to work with our listwise LTR model (Sections 3.3 and 3.4).
- (4) We extensively evaluate our LTR approach against other ML models as well as SQL server which includes a well-tuned cost model. Our results show that our LTR approach matches the performance of a commercial database's optimizer for short-running queries and even outperforms it (up to 5×) for long-running queries. Our approach consistently

outperforms state-of-the-art ML-based baselines: it chooses plans with up to an order of magnitude better runtime performance.

We conclude the paper with related work and a discussion on future directions.

## 2 Preliminaries

Learning-to-rank (LTR) algorithms are a special type of supervised machine learning – the task of ranking is neither a classification nor a regression task. Instead, the goal of LTR algorithms is to predict a score for a set of items with the goal of sorting them by their predicted score such that a ranking can be given as a result [Li11]. An LTR algorithm considers an item with a high score to be more relevant to a given query than an item with a low score. Consequently, the goal of LTR tasks is different from classification or regression tasks and the algorithm itself does not regard the predicted score but the correct order of the input items for calculating the loss and improving the model. Similarly, the metrics for measuring predictive performance also differ from the “standard” metrics used for classification or regression tasks.

Generally, there are three main approaches for LTR algorithms: pointwise, pairwise, and listwise [Li11]. Pointwise algorithms handle each item separately as input and calculate its relevance score, which in turn is used to sort all items and get the final ranking. However, these algorithms do not consider the inter-dependency between items as the loss is calculated separately for each item. Unlike pointwise approaches, in a pairwise approach two items serve as input for one prediction which is given by three distinct values  $\{-1, 0, 1\}$ . The model outputs  $-1$  when it predicts that the first input sample has a lower rank than the second input sample; otherwise, it outputs  $1$ . The result is  $0$  if the two inputs are the same or if the model considers them to have the same relevance. In the listwise algorithms, the entire list of items belonging to the same query is used as input and the output is a corresponding sorted list. Both pairwise and listwise algorithms belonging can often be further categorized based on their underlying ML algorithms, such as Support Vector Machines (SVM) or Neural Networks (NN). As we will detail in the next section, we chose a listwise approach, and in particular the LambdaLoss [Wa18], which uses the idea that the scores define a distribution:

$$L(y, s) = - \sum_{y_i > y_j} \log_2 \sum_{\pi} \left( \frac{1}{1 + e^{-\sigma(s_j - s_i)}} \right)^{\delta_{ij} |G_i - G_j|} H(\pi|s) \quad (1)$$

where  $s_i$  is the true score for item  $i$ , and  $y_i$  describes the weight, i.e., the relevance grade the model predicts,  $H(\pi|s)$  is a “hard assignment distribution” [Wa18] with  $H(\hat{\pi}|s) = 1$  and  $H(\pi|s) = 0$  for all permutations of the items with  $\pi \neq \hat{\pi}$ . Thereby,  $\hat{\pi}$  is the permutation in which all documents are sorted correctly.  $\delta_{ij}$  a tunable parameter and  $G$  is the gain function  $2^y - 1$ . Furthermore, the researchers extended this loss function to a top- $k$  approach, similar to ListNet, with  $k$  being a tunable parameter. The authors define this extended loss such that



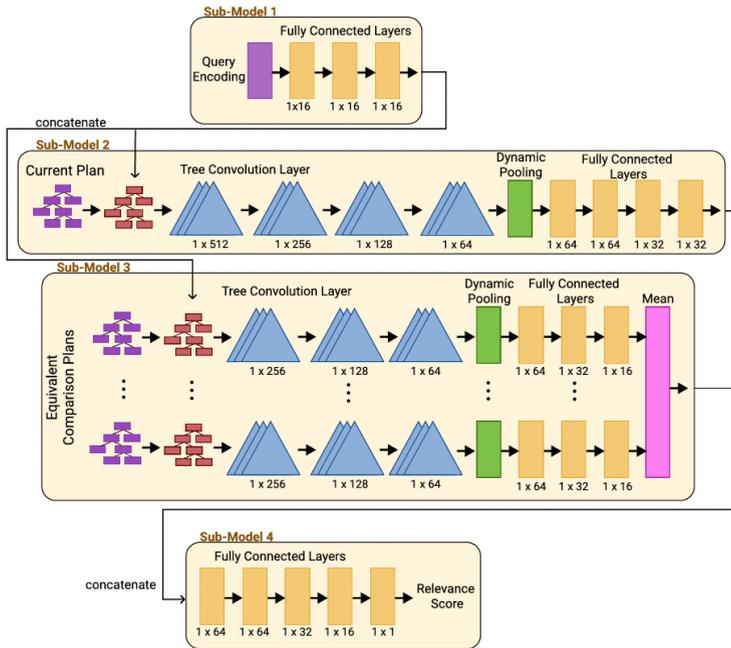


Abb. 2: Neural network architecture of our LTR model.

approaches, pairwise approaches, and listwise approaches [Li11]. Pointwise algorithms do “not consider the inter-dependency between items” [Li11] as the loss is calculated separately for each item. In our case, this means that the rank of an execution plan would be solely based on its (estimated) runtime and would not consider any comparison metric among equivalent plans. This would defeat the purpose of using an LTR model in the first place. For this reason, we discard a pointwise approach. Because traditional plan enumeration algorithms prune the search space by comparing two subplans at a time, it is intuitive to use a pairwise approach for our LTR model. However, such an approach (i) uses an objective loss for minimizing errors in classification of pairs rather than minimizing errors in ranking of items, (ii) requires a large number of pairs to train on, which can be computationally costly to generate, and, (iii) assumes that the pairs are generated i.i.d. which is not true in the case of equivalent execution plans. *We, thus, aim for a listwise approach.* Our experimental results shown in Section 4.3 demonstrate how our proposed listwise solution outperforms a pairwise approach.

Once we settle for the LTR approach, the next problem that we need to tackle deals with the neural network architecture that we shall use to build an LTR model. We get inspired by a popular listwise LTR architecture called FATE [PGH18], and the architecture of [Ma19] that uses a neural network for estimating the runtime of execution plans. Figure 2 depicts

the neural network architecture of our LTR model. Similarly to FATE, our model uses equivalent execution plans as additional information to the current plan, for which the model has to predict a relevance score. Consequently, the model expects multiple equivalent plans as input and estimates the relevance score for each one of them. The model considers every plan of the inserted list individually as the current plan and utilizes every other plan in the list as a comparison plan.

For a given query (either during prediction or training), we first calculate the query encoding and plan encoding, both depicted with violet color in Figure 2. The query encoding forms a vector while the plan encoding is tree-shaped (see Section 3.3). Then, a first sub-model (Sub-Model 1), consisting of several fully-connected layers, takes the query encoding as input and outputs a vector of length 16. Subsequently, the model concatenates the output of Sub-Model 1 with the plan encodings of all plans, i.e., the current plan and the comparison plans. The extended plan encoding of the current plan serves as input for a second sub-model (Sub-Model 2) while all extended comparison plans are inserted into a third sub-model (Sub-Model 3) separately. Both mentioned sub-models have a similar architecture consisting of multiple tree convolution layers, a dynamic pooling layer, and several fully-connected layers. The difference between the two sub-models is the number of layers as (Sub-Model 2) consists of more layers. The reason for this is that we want to put more focus on the current plan than on the comparison plans. Sub-Model 3 consists of a final layer that calculates the mean of the outputs of the comparison plans. Before the model transfers this vector to the last sub-model (Sub-Model 4), it concatenates this resulting mean vector and the vector produced by (Sub-Model 2) so that the final model considers each plan in comparison with the rest. Finally, it inserts the result into (Sub-Model 4), consisting of five fully-connected layers. The last fully-connected layer outputs the relevance score.

Note that the model architecture itself already implies a listwise comparison strategy regardless of the loss functions used. To further enforce the listwise approach we use the LambdaLoss [Wal8] function, which incorporates a ranking metric, as described in Section 2. Note that our main focus is not the implementation of a new loss function, which is why we choose to use an existing one.

### 3.2 Scoring of execution plans

The required training data to build an LTR model include execution plans (data points after featurization) and their rank (label). Yet, the data we usually have available from execution logs is execution plans and their runtimes.<sup>¶</sup> One may think that it suffices to simply sort the plans and based on their runtime and convert it to a rank. However, this is not ideal because we lose information on how the runtimes differ among the plans. For example, in the case of OLAP queries, if plan A has a runtime of 200ms and plan B has a runtime of 205ms,

---

<sup>¶</sup>The way to acquire such data is orthogonal to our approach and we thus, do not discuss it in the paper. For our implementation we have used DataFarm [Ve21].

we would like to give them the same score as their difference in runtime is insignificant. Conversely, in the case of OLTP workloads we would like to assign a different score to the above two plans. In few words, the goal of using a score function is to take into account the relative performance among the plans.

To enable the model to learn relevance scores to rank different plans, we need to devise a score function for scoring the training plans. The intuition behind devising such a score function is that, based on the runtimes of the plans, we want to teach the model which plans are good and bad, but also which plans have a similar performance. The best plan for a specific query should receive the highest relevance score, while the worst plan should receive a low score. Additionally, we want to be able to map plans with a similar runtime to the same score. However, the definition of “plans with a similar performance” can vary from query to query when looking only at execution times. Depending on the query, similar plans can differ, for example, in their run time by only  $1ms$  or by  $2000ms$ . Utilizing a score function ensures that the plans follow a uniform scoring scheme. We devise two different score functions: a linear score function and an advanced global score function utilizing agglomerative clustering. For both functions, we use a hyperparameter  $s_{max}$  that determines the maximum score of an execution plan. Thus, by tuning  $s_{max}$  we can adjust the granularity of the score assignment, depending on our scenario (i.e., OLAP vs OLTP cases).

### 3.2.1 Linear score function

Our first proposed score function is the *linear score function*, which is a straightforward approach for transforming the increasing run times into decreasing relevance scores. The function uses a decreasing linear function  $f : \mathbb{N} \rightarrow \mathbb{R}$  for calculating the scores such that the larger the execution time for a specific plan, the smaller the relevance score becomes. To calculate a score, the function maps an execution time in ms to a real-valued score  $[0, s_{max}]$  with respect to other equivalent plans for the same query and the hyper-parameter  $s_{max}$  defining the maximum score. The parameter  $s_{max}$  influences the upper bound beyond which the values are receiving a score of 0, which we explain in the following.

Considering one query at a time, the linear score function calculates a linear function between the minimum run time  $t_{min}$  of a set of equivalent plans and  $s_{max} \times t_{min}$ . This minimum value  $t_{min}$  receives a score of  $s_{max}$ , and the value  $s_{max} \times t_{min}$  gets mapped to a value of 0. Between those two values, the score decreases linearly. Additionally, the function sets every execution time higher than  $s_{max} \times t_{min}$  to 0. Equation 2 expresses this formula.

$$f_{s_{max}}(x_i) = \begin{cases} s_{max}, & \text{if } x_i = t_{min} \\ \frac{-s_{max}}{(s_{max}-1)} \left( \frac{x_i}{t_{min}} - s_{max} \right), & \text{if } t_{min} \leq x_i \leq t_{min} \times s_{max} \\ 0, & \text{if } x_i \geq t_{min} \times s_{max} \end{cases} \quad (2)$$

|      |                  |                |            |                  |           |            |            |      |                |
|------|------------------|----------------|------------|------------------|-----------|------------|------------|------|----------------|
| Sort | Stream Aggregate | Hash Aggregate | Merge Join | Nested Loop Join | Hash Join | Index Scan | Table Scan | Null | Estimated Rows |
|------|------------------|----------------|------------|------------------|-----------|------------|------------|------|----------------|

Abb. 3: Features used in the plan encoding.

To clarify this score function, we consider the following example for the linear score function with  $s_{max} = 5$ . For a query, the best plan of all possible plans, i.e., the plan with the shortest run time, needs 10s for its execution. Therefore, the function sets  $t_{min} = 10$  and maps this plan to a score of 5. Every plan with a run time higher than or equal to 50 s ( $s_{max} \times t_{min} = 5 \times 10$ ) receives a score of 0. Between these two runtimes, the score decreases linearly, e.g., a query with a run time of 25s receives a score of 2.5 and a run time of 20s receives a score of 3.75.

### 3.2.2 Global agglomerative score function

Besides the linear score function, we devise a second score function, the *global agglomerative score function*. This function utilizes the unsupervised clustering algorithm *agglomerative clustering* [SB13]. Unlike the linear score function, it considers all available plans with their execution time at once, independently of the query they belong to. To make a global scoring feasible, the score function scales the query plans at first. For each query separately, it takes the minimum execution time and divides every equivalent plan for this respective query by the minimum execution time of this query. This results in factors in  $[1, +\infty)$  describing how many times slower the execution of this plan is w.r.t. the minimum execution time for this query. During calculation, all factors for all queries of the training set are stored in an array, and on this array, we apply agglomerative clustering. However, before the clustering step, we need to remove outliers. This is because a huge outlier, i.e., a plan with long execution time, could be clustered as its own cluster while plans with small values can be all assigned into one cluster due to the nature of the agglomerative clustering algorithm. To achieve this, the function calculates the *border*-th percentile of factors, where *border* is a hyperparameter. The algorithm sets every factor below a percentile *border* to the value of *border*.

After execution of the agglomerative clustering, we sort the resulting clusters by the minimum value for each cluster. Finally, all query plans within the cluster with the smallest value receive a value of  $s_{max}$ , and for every next cluster in the sorted version, the scores attached decrements by 1. We show the explained procedure in Algorithm 1.

### 3.3 Featurization of execution plans

As described in the previous subsection, our model architecture requires two different inputs: the query encoding and the plan encoding. The query encoding is a simple vector

**Input:**  $s_{max} \in \mathbb{N}$ ,  $border \in (0, 1]$  and execution times  $T = \{T_{11}, \dots, T_{nq}\}$  with  $q$  being the query number and  $n$  being its index inside query  $q$

**Output:** Scores  $S = \{S_{11}, \dots, S_{nq}\}$

Create empty array  $factors$

```

for  $i \in [1, q]$  do
  minimum = min( $\{T_{1i}, \dots, T_{ni}\}$ )
  for  $t$  in  $\{T_{1i}, \dots, T_{ni}\}$  do
    | Append  $\frac{t}{minimum}$  to factors
  end

```

**end**

percentile = get  $border$ -percentile of factors

**for**  $f$  in factors where  $f \geq percentile$  **do**

|  $f = percentile$

**end**

AgglomerativeClustering(factors, numberClusters= $s_{max}$ )

Sort clusters by their minimum value

Give first cluster a score of  $s_{max}$  and decrement  $s_{max}$  for succeeding clusters

**Algorithm 1:** Calculation of the global agglomerative score

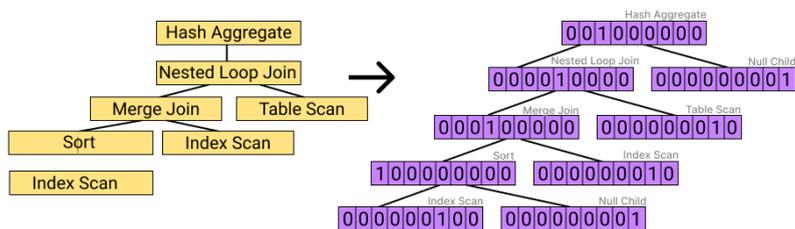


Abb. 4: Example featurization of an execution plan (plan encoding). For the sake of simplicity, we omit the last feature, which is the estimated intermediate cardinalities.

describing several different query features, such as whether a GroupBy exists or how many joins the query contains. The plan encoding consists of multiple vectors arranged in a tree shape describing the utilized operators of the execution plan at each position by using a one-hot-encoding strategy. The features of the plan encoding are similar to the features used in [Ma21]; we encode every operator to a one-hot-encoded vector for every important node in the plan. Thereby, we consider the different join and aggregate physical operators as well as two scan operators – index scan and table scan – and the sort operator. Figure 3 shows the features we use for the one-hot-encoding of each operator in the plan encoding.

The tree-shaped plan encoding considers binary trees which is the most popular type of query plans. As the tree convolution layer requires each inner node of the tree to have two children, we need to “fill” the children of unary operators, such as sort or aggregate. For this, we use a feature vector “Null” to indicate “null-children” and add a second “null-child”

|              |              |                 |                          |               |               |
|--------------|--------------|-----------------|--------------------------|---------------|---------------|
| Has Order By | Has Group By | Number of Joins | Estimated Number of Rows | Max. Relation | Min. Relation |
|--------------|--------------|-----------------|--------------------------|---------------|---------------|

Abb. 5: Example featurization of an execution plan (plan encoding).

to unary operators. In addition to the one-hot-encoding of the operator, we extract the estimated resulting rows from the database optimizer and append this estimation to our feature vector. Figure 4 illustrates an example featurization of an execution plan. The plan in this example consists of two joins, three scans, a sort, and an aggregate. Moreover, we can see the use of null-children, which are inserted as the right child for the sort and the aggregate node. In the figure, we omit the estimated rows feature for simplicity.

In addition to the plan encoding, we have to encode the query into a feature vector as illustrated in Figure 5. In contrast to [Ma21], we keep this vector simple and independent of the relations used, i.e., there is no one-hot-encoded information about the utilized columns or relations. This allows our featurization to be used out of the box for any dataset. Instead, it includes information about whether the query utilizes a GroupBy and an OrderBy as well as the number of joins and the result size of the query estimated by the database. Furthermore, we append information about the maximum and the minimum number of rows of the relations used.

Before training, we normalize all features of the plan and query encoding. Therefore, for every feature used, we extract its maximum and minimum value and normalize all the features using the formula:  $norm(x) = \frac{(x-min)}{(min-max)}$ .

### 3.4 LTR-based Plan Enumeration

Bottom-up enumeration algorithms are very popular with existing database systems. In a learning-based query optimizer, these algorithms are responsible for calling the model and receiving its cost estimations (see Figure 1). However, they utilize a pairwise comparison of plans during the enumeration. This can work out of the box with an ML model that simply outputs the estimated runtime of a plan or with a pairwise LTR model. However, our model is listwise, i.e., it outputs a ranked *list* of plans. One could use it with the default plan enumeration if the list consists of only two plans. However, this would be inefficient. For this reason we need to adapt the state-of-the-art enumeration algorithms to work with ranked lists of plans and not with the cost of pairs of plans.

After considering different bottom-up enumeration algorithms, we decide to use the DPccp algorithm by Moerkotte and Neumann [MN06]. This algorithm has the benefit of using graph algorithms, making it a better fit to transform the graph into the tree-structured plan-level encoding, which is needed for the LTR model to estimate the relevance score. To make DPccp work with our model we need to adjust the part of the algorithm that compares two plans at a time w.r.t. their cost. As a result, the decision for the best sub-plan in a set of

equivalent sub-plans is made shortly before the sub-plan is needed for joining purposes. Furthermore, we extended the algorithm to get a third input, parameter  $k$ , next to the query graph  $G$  and model  $model$ . Parameter  $k$  is tunable and decides to how many execution plans the model reduces the list of possible plans during its prediction. For example, for  $k = 10$ , at every prediction of the model, the model returns the top  $k = 10$  plans with the highest predicted relevance score. Consequently, the enumeration algorithm calls the model whenever the number of possible equivalent plans exceeds  $k$ .

**Input:** A connected query graph  $G$  with relations  $R = \{R_0, \dots, R_{n-1}\}$ , LTR model  $model$  and parameter  $k$

**Output:** an optimal bushy join tree

```

for all  $R_i \in R$  do
  | PossiblePlans( $\{R_i\}$ ) = getScan( $R_i$ );
end
for every  $S_1$  in EnumerateCsg( $G$ ) do do
  | for every  $S_2$  in EnumerateCmp( $G, S_1$ ) with  $S = S_1 \cup S_2$  do
  | | if length of PossiblePlans( $S_1$ ) >  $k$  then
  | | |  $p_1 = model.predictBestK(PossiblePlans(S_1), k=k)$ 
  | | | end
  | | | else
  | | | |  $p_1 = PossiblePlans(S_1)$ 
  | | | | end
  | | | if length of PossiblePlans( $S_2$ ) >  $k$  then
  | | | |  $p_2 = model.predictBestK(PossiblePlans(S_2), k=k)$ 
  | | | | end
  | | | | else
  | | | | |  $p_2 = PossiblePlans(S_2)$ 
  | | | | | end
  | | | | Append plans from CreateJoinTrees( $p_1, p_2$ ) to PossiblePlans( $S$ )
  | | | | end
  | | | end
  | | end
  | end
end
end
return  $model.predictBestK(PossiblePlans(R), k=1)$ 

```

**Algorithm 2:** Adjusted DPccp algorithm based on [MN06]

Algorithm 2 shows the pseudocode of our adapted enumeration algorithm. In the beginning, the algorithm starts a for-loop for receiving all possible scans for each node in  $G$ , similar to the first loop in the original DPccp. After it has calculated every possible scan and saved it in a data structure *PossiblePlans*, it starts a second and third for-loop for calling the sub-graphs with two functions – *EnumerateCsg* and *EnumerateCmp* – that we extracted from the original DPccp algorithm and left unchanged. Inside the inner for-loop, for both sub-graphs  $S_1$  and  $S_2$ , the algorithm tests if the number of possible plans for calculating the respective subgraph exceeds  $k$ . If it does, the algorithm calls the LTR model, reducing the number of possible plans to  $k$  plans for which it predicts the highest relevance scores. Afterward, it stores the remaining  $k$  plans in variable  $p_1$  or  $p_2$ . If the number of possible plans is already smaller or equal to  $k$ , the algorithm directly stores all plans in  $p_1$  or  $p_2$  without further reduction. After  $p_1$  and  $p_2$  are created, the function *CreateJoinTrees* calculates all possible joins for each pair in  $p_1$  and  $p_2$ , i.e., it considers every join operation. The algorithm stores the resulting sub-execution plans for  $S$  with  $S = S_1 \cup S_2$  in our data

structure  $PossiblePlans(S)$ . Eventually, when both for-loops have finished, the model predicts the best plan from  $PossiblePlans(R)$ , i.e., for the entire graph, with  $k = 1$ .

## 4 Experimental Evaluation

We evaluate our proposed solution against the cost-based query optimizer of a commercial database system and two state-of-the-art learning-based query optimizers. Specifically, we evaluate whether the query performance for a well-known query workload improves with our learning-to-rank approach compared to other baselines for both seen and unseen data.

### 4.1 Setup

We first describe our experimental setup, which includes our utilized hardware/software, baselines, and characterization of the utilized training and test data.

**Hardware and Software Specification.** We used a computer with an AMD Ryzen 7 1700X Eight-Core processor at 3.77 GHz with 32 GB main memory. We implemented all our components in Python version 3.9.7 and used PyTorch 1.10.0 for building both ours and the comparison ML models. For fair comparison, we use a commercial database to execute the plans chosen in of the cases we study. To automate the execution of our execution plans in the database system, we used the Python library pyodbc version 4.0.32.

**Baselines.** We use a commercial database (denoted as DB  $X$ ) and three different comparison models as baselines. The latter are plugged in our implementation that uses the adapted enumeration algorithm as described in Section 3.4. We compare against two state-of-the-art models of learning-based optimizers – Bao [Ma21] and Neo [Ma19]<sup>||</sup> – as well as a simple baseline comparison model. This baseline model is a pairwise LTR approach and has a similar architecture to Bao’s model. It uses the linear score function with  $s_{max} = 50$  and RankNet [Bu10] as a loss function for training. For enumerating through the plans, we use our adapted DPccp enumeration algorithm for every case and set  $k = 10$ . For our approach (denoted as LTR model), we have ran extensive experiments to find the best combination of score functions and hyperparameters as well as loss functions. However, due to space limitations, we report the performance of our best model which uses the global agglomerative score function with  $s_{max} = 50$  and  $border = 97$  and the LambdaLoss function with the top-k extension (see Section 2).

**Training Data.** For training, testing and evaluating our solution, we decide to mainly use the database schema from the TPC-H benchmark<sup>\*\*</sup>. Due to the lack of already labeled training data fitting TPC-H’s database scheme, we generate and label appropriate training data

---

<sup>||</sup> In the future, we plan to experiment with zero-shot cost models [HB22] as well.

<sup>\*\*</sup> See: <https://www.tpc.org/tpch/>

Tab. 1: Number of generated testing queries based on their number of joins and runtime on 1GB data.

| Query Length   | 1 Join | 2 Joins | 3 Joins | 4 Joins | 5 Joins | 6 Joins | 7 Joins |
|--|--------|---------|---------|---------|---------|---------|---------|
| Short Queries (< 2sec)                                   | 8      | 8       | 8       | 8       | 8       | 8       | 8       |
| Medium Queries ( $\geq 2\text{sec}$ & $< 30\text{sec}$ ) | 2      | 8       | 8       | 8       | 8       | 8       | 8       |
| Long Queries $\geq 30\text{sec}$                         | 0      | 2       | 4       | 7       | 8       | 8       | 5       |

using DataFarm [Ve21], a training data generation tool for ML-based query optimization. DataFarm uses as input a small amount of SQL queries and information about the different tables to create further similar queries. The initial version of the tool was implemented for Flink and, thus, we had to adapt some parts for our purpose. Specifically, we use six different TPC-H queries serve as input (namely, Q1, Q3, Q11, Q13, Q17, and Q21). These queries contain operators that our featurization process supports. Based on these six queries, we let DataFarm generate 2,000 SQL queries with maximum seven joins and five variations each. However, to get our training data we need runtimes for specific execution plans and not only for SQL queries. Thus, we insert each query into the plan enumeration algorithm and receive different valid execution plans. Because the amount of possible plans increases exponentially with the amount of relations in the query, we decided to implement a limit of 100 different execution plans per query. Thereby, the enumeration algorithm prunes the sub-plans randomly such that the resulting execution plans differ for different queries. Lastly, we need the runtime of these execution plans. Therefore, we let DB X run the different execution plans on 1GB TPC-H, forcing the usage of a specific plan. Due to time limits, we are not able to label all generated plans, resulting in a training set consisting of 25,067 plans. Furthermore, we set the timeout for all queries to six hours, which results in the problem that not all execution plans have an execution time in the end. For our LTR model, this does not pose a problem, as plans with a bad execution time are always assigned a relevance score of 0. However, for the two comparison models, Bao and Neo, this circumstance is a problem, as these models need the execution time of the plans for their estimations. To solve this problem, we set these estimations to 6 hours.

**Testing Data.** For evaluating the results of the different models, we decide to use several different queries. The first set of testing queries consists of 136 queries which we generated during the training data generation process but did not use for training purposes. We distinguish these queries based on their runtime: *short* queries with a run time of less than 2sec, *medium* queries with a run time of less than 30sec but more than 2sec, and *long* queries needing more than 30sec. For every model tested, we let it predict the best execution plan for each query with the help of our plan enumeration algorithm. Afterward, we insert this plan into the commercial database to measure its real execution time. In Table 1, we show the exact number of queries split by their number of joins for each category.

To test whether the performance of our model’s suggested execution plan remains similar when increasing the size of the database, we increase the size of the data to 5GB and let our model predict the same test queries but using the larger dataset. However, due to time limitations, we had to reduce the number of queries from 136 to 95. As many queries have longer execution times on the larger dataset, we do not use the same definition of short,

Tab. 2: Number of generated testing queries based on their number of joins and runtime on 5GB data.

| Query Length   | 1 Join | 2 Joins | 3 Joins | 4 Joins | 5 Joins | 6 Joins | 7 Joins |
|----------------|--------|---------|---------|---------|---------|---------|---------|
| Short Queries  | 8      | 8       | 7       | 7       | 6       | 2       | 0       |
| Medium Queries | 2      | 5       | 4       | 7       | 4       | 2       | 3       |
| Long Queries   | 0      | 5       | 6       | 4       | 9       | 3       | 3       |

medium and long queries. Every query with a runtime with 15 seconds or less is regarded as short query. Medium queries have an execution time between 15 and 60 seconds, and we consider every query with more than a minute execution time as long query. Table 2 shows the corresponding queries distribution.

Lastly, to evaluate the performance of our models on *unseen data*, we use the IMDB dataset in combination with queries from the Join Order Benchmark (JOB)<sup>††</sup>. Out of these queries, we use 26 different queries all of them having between 3 and 11 joins for our experiments. Note that the TPC-H data we use for training has a maximum of 7 joins, and, using this dataset, we can test our model’s performance for queries with a larger number of joins.

**Evaluation Metrics.** For each query  $q$  we measure the execution time of the models’ best predicted plan when inserted in DB X. Furthermore, we record the time the commercial database, DB X, needs for running  $q$ . To ensure a fair comparison, we let DB X at first output the execution plan for  $q$  and then insert it again into DB X as we do for the other models and measure its runtime. For our evaluation, DB X’s time serves as a baseline as it contains a highly tuned cost-based optimizer. We, thus, use not only the exact execution times but also the ratio of the execution time of the models’ predicted plans to DB X baseline, i.e., we divide for every query the time of our models’ plans by the time that DB X needs.

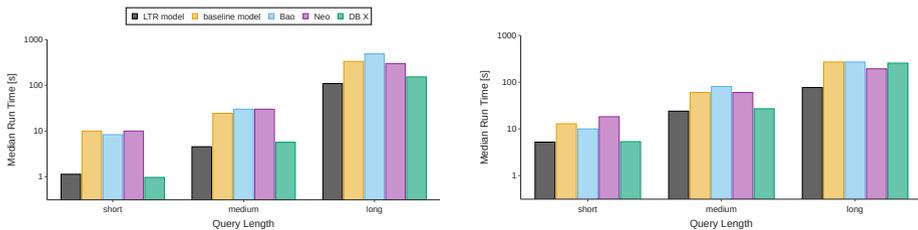
## 4.2 Performance against State-of-the-Art

We first evaluate how the models perform on our generated test queries.

### 4.2.1 Absolute performance

For a first overview of the model’s performance, we plot the median execution time for the predicted plans categorized by the query type in Figure 6a. Note that the  $y$  – axis is in logarithmic scale. While our LTR model seems to have a good performance which is able to match DB X’s performance, the plans of the comparison models have an extremely poor performance. To investigate the reasons for their bad performance, we analyze some of their predicted execution plans. In examining some of the plans, we note that Neo often predicts merge and nested loop joins for queries where the other models predict mainly hash joins.

<sup>††</sup><https://github.com/gregrahn/join-order-benchmark>



(a) 1GB data – used in training data generation–

(b) 5GB data – not used in training data generation–

Abb. 6: Median runtime of our LTR model, the three comparison models, and DB X. Our LTR model matches the performance of DB X and exceeds it for long-running queries while it always outperforms Neo, Bao, and the baseline model.

Most of these predicted merge joins require a preceding sort. We hypothesize that this is the reason why the plans predicted by Neo often throw a timeout. When analyzing the plans of the baseline model and Bao, we observe that also these models often predict merge joins with a preceding sort even though not as often as Neo does. Similar to Neo, we assume that these join operators in combination with sorts result in worse plans.

Note that we trained our model on execution plans with runtimes retrieved over a database with size of 1GB. Therefore, it is possible that our model has a different performance on queries for larger datasets. To test this, we run another set of experiments on 5GB of data for our model, the three comparison models and DB X. Here, we again use the same generated queries as before even though we had to reduce the number of test queries to 95 queries (see Table 2). We show the median runtimes of the different query types in Figure 6b. Similar to the prediction on the smaller dataset, our model significantly outperforms the comparison models. For short and medium queries, it matches SQL Server’s performance, and it is also able to outperform SQL Server for long queries.

### 4.2.2 Relative performance

To gain more insights on the results of the predicted plans, we plot the relative performance of the models’ chosen plans in relation to DB X. Here, we divide for every predicted plan its time by the time DB X needed. Figure 7 shows the resulting boxplots when the 1GB TPC-H dataset is loaded in DB X. A value below the red line signals that the corresponding model predicted a plan with a runtime lower than the DB X suggestion. In the boxplots, we can see clearly that our model has the best performance of all four models. Furthermore, except for some outliers, it is often able to match or outperform DB X, especially for medium and long queries. For short queries, it is worse than DB X. However, since short queries need only 2sec for execution on DB X, we can see that for most queries our model’s plan is only

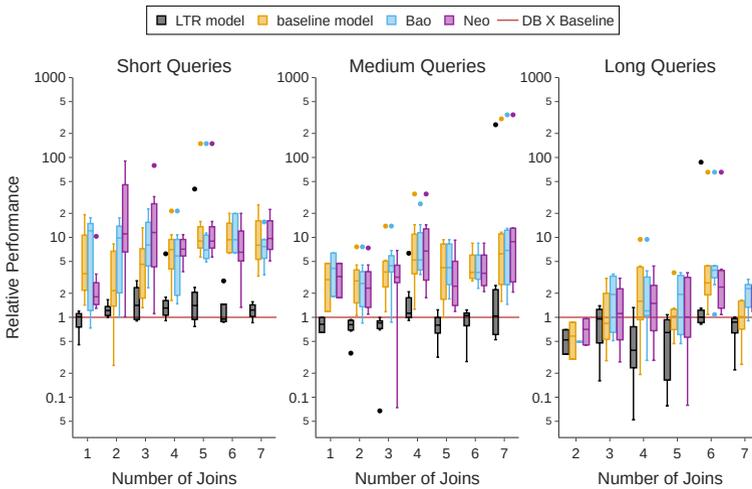


Abb. 7: Relative performance of our LTR model and the comparison models in relation to DB X. Values  $> 1$  denote worse performance compared to DB X while values  $< 1$  denote better performance.

slightly slower, i.e., the time difference lays within a second, while for the long queries our model can be 5 hours faster (4,015sec for LTR model compared to 22,292sec for DB X).

Similarly, Figure 8 shows the boxplots for 5GB of TPC-H data, while using 1GB for training. These plots emphasize that for most long-running queries and for many medium ones, our model is able to outperform DB X. The two queries for which our model's plans are significantly faster than DB X's plans are a query with performance difference of 4,857sec and 1,467sec, respectively. For most queries where our model's plan is faster, DB X utilizes a stream aggregate with a pre-appended sort which slow down the query runtime.

### 4.3 Performance on Unseen Data

The goal of our solution is also to work with unseen data. To validate this, we load the IMDB dataset into DB X and execute 26 different queries, extracted from the same dataset. Again, we evaluate our model against Bao, the baseline model and DB X. Due to the fact that Neo's query featurization is tailored to the TPC-H dataset, we cannot use it for comparison. For the other models and DB X, we measure the plans runtime shown in Figure 9.

We observe that our LTR model and DB X have very similar performance for most queries, such as *14a* or *3a*. This is an astonishing result per se as our LTR model requires no manual tuning compared to the hours of human-engineered cost model of DB X. For some queries, such as *17a* and *31a*, our LTR model even manages to outperform DB X and achieve up to  $5\times$  better query performance. When analyzing our model's predicted plans, we observed

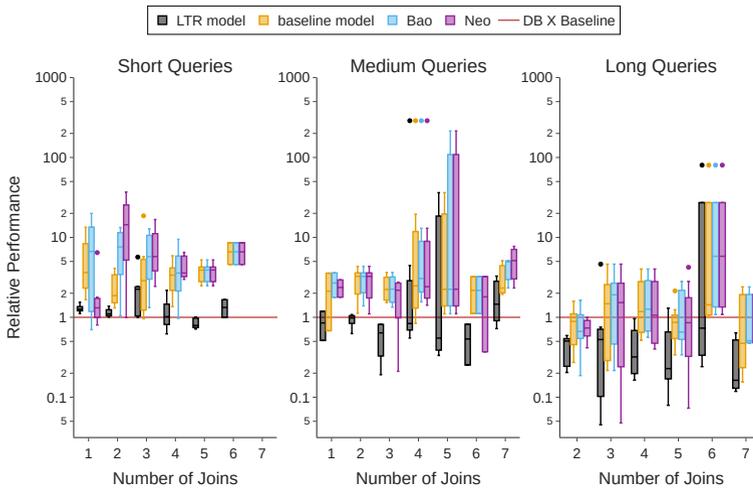


Abb. 8: Relative performance of queries on 5GB of data for our model and the comparison models to DB X. Our LTR model performs the best among all four models.

that our model predicts only hash joins. We assume that this might result from the nature of the IMDB data. In contrast to the data from the TPC-H benchmark, this dataset does not contain any indexes referencing to other tables which can be used to sort the table while scanning. Therefore, a merge join would always require at least one sort. We hypothesize that these sorts let the model predict a worse relevance score for queries with a merge join. Nevertheless, it seems that our model overfits slightly in favor of hash joins because it also does not predict any nested loop joins. However, in some cases DB X outperforms our model, e.g., for queries *20a* and *26a*. An analysis of the corresponding plans shows that DB X utilizes adaptive joins for these queries, a join algorithm we did not use while training because we were not able to produce valid plans with this join type. We, thus, consider the integration of adaptive joins future work. In addition, we observe that LTR performed slightly better than the baseline for most queries and for some of them (e.g., queries *17a*, *5a*, *6a*) it was significantly better. Interestingly, we also observed that Bao predicts plans with a bad performance for almost every query. Looking into its predicted execution plans for queries where Bao timeouts, we realized that it always predicts bushy join trees with nested loop joins as well as many merge joins with preceding sorts for both relations. For most queries, this seems to be a bad combination.

In conclusion, our model’s performance on unseen data is good even though it only predicts hash joins. We assume that including data of queries for other datasets as well as adaptive joins might solve its overfitting issue. Nevertheless, our model is able to predict proper execution plans on unknown data, and it predicts some better plans than DB X resulting to a performance increase up to 5×, while it does not require manual tuning.

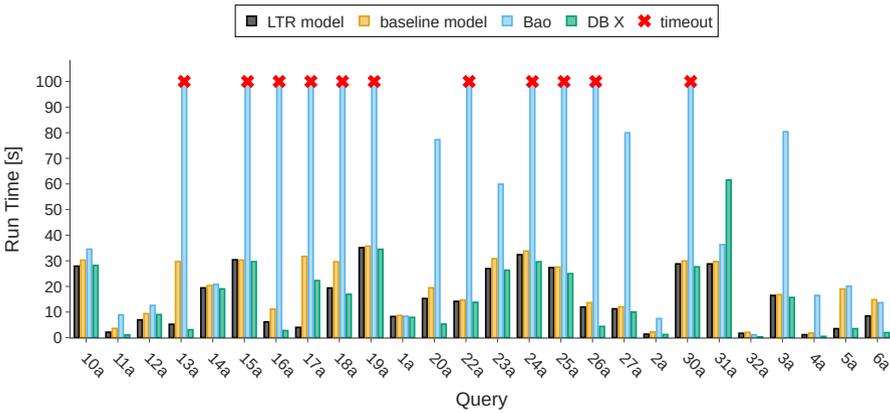


Abb. 9: Rutimes of chosen plans using the unseen IMDB dataset and JOB query benchmar.

## 5 Related Work

To our knowledge there is no work that leverages an LTR approach for query optimization. Works aiming at replacing a cost model with an ML model focus on estimating the execution time of plans, typically using regression models. Similarly, there are several efforts using ML for query performance prediction.

Early works that incorporated the use of ML for query performance prediction, mainly use a set of models, e.g., one for each operator, and aggregate them into a single cost. For example, [Ak12] uses different granularities for encoding and estimating plans, such as on plan-level, operator-level, and a hybrid approach. Compared to our solution, they not only use a different encoding strategy, but also different models for the plan-level and operator-level and not one model for the whole process.

A more recent work [MP19] for predicting query performance proposes the use of a modular principle to combine NNs with two hidden layers. Depending on the operator, the model utilizes a different neural network unit, whereas every unit produces an output with a similar structure to ensure consistency. The resulting deep NN represents the tree structure of the plan. The main difference to our solution is that the tree structure lies inside the model while we use it in the feature vector. Furthermore, the model building requires the execution time of every operator on its own, which leads to a more elaborated training data collection.

Neo [Ma19], a learned query optimizer, includes a model architecture for estimating the execution time of plans with NNs and uses a reinforcement approach for the join ordering. Their NN also uses tree convolution layers combined with a query vector, similar to our model approach, even though they do not use equivalent execution plans for their estimation. Another difference to our approach is that they do not inspect other important operators like

aggregation or sort, only joins and scans and their featurization scheme is tight to the input dataset schema. Bao [Ma21], another learned query optimizer, uses a NN combined with reinforcement learning to predict the cost of an execution plan. However, the estimated cost is not used for selecting the best plan but for providing query hints to the database. While the NN layers are similar to our neural network, the featurization and the architecture of the model are different. In their features, they use no query encoding, which is also depicted in the architecture. Importantly, they do not use comparison queries during prediction.

[Ka20] proposes the use of ML, in particular a regression model, in the plan enumeration for predicting plan performance for the cross-platform system, Apache Wayang (incubating) (former Rheem) This work differs from ours not only because they use a vector representation and not a tree-based representation for the featurization but also because their focus is on performing the plan enumeration using entirely vectors.

## 6 Conclusions

We presented a learning-to-rank (LTR) approach for ML-based query optimization. Our proposal consists of (i) a novel listwise neural network architecture that considers not only one plan at a time but also its equivalent plans and a ranking-based loss function, (ii) two score functions that transform the runtime of the execution plans to a ranking score used as label, (iii) a featurization scheme for the execution plans that covers a wide range of operators and is oblivious to the dataset, and (iv) an adaptation of a bottom-up enumeration algorithm to work with our listwise LTR model. We evaluated our approach with a commercial database that includes a cost-based optimizer and two state-of-the-art regression-based ML approaches. Our results show that our LTR approach was able to match the performance of the commercial database for short and medium queries, while it outperformed it for long-running queries (up to 5×, leading to a runtime decrease of up to 5h). In addition, our approach systematically chooses better plans than state-of-the-art learning-based approaches, resulting in a performance speedup of over an order of magnitude.

## Acknowledgments

This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

## Literaturverzeichnis

- [Ak12] Akdere, Mert; Çetintemel, Ugur; Riondato, Matteo; Upfal, Eli; Zdonik, Stanley B.: Learning-based Query Performance Modeling and Prediction. In: ICDE. S. 390–401, 2012.
- [Bu10] Burges, Christopher J.C.: From RankNet to LambdaRank to LambdaMART: An Overview. Learning, 2010.

- [HB22] Hilprecht, Benjamin; Binnig, Carsten: Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *Proc. VLDB Endow.*, 15(11):2361–2374, 2022.
- [Ka20] Kaoudi, Zoi; Quiané-Ruiz, Jorge-Arnulfo; Contreras-Rojas, Bertty; Pardo-Meza, Rodrigo; Troudi, Anis; Chawla, Sanjay: ML-based Cross-Platform Query Optimization. S. 1489–1500, 2020.
- [Ki19] Kipf, Andreas; Kipf, Thomas; Radke, Bernhard; Leis, Viktor; Boncz, Peter; Kemper, Alfons: Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In: *CIDR*. 2019.
- [Li11] Liu, Tie-Yan: *Learning to Rank for Information Retrieval*. Springer-Verlag, 2011.
- [Ma19] Marcus, Ryan; Negi, Parimarjan; Mao, Hongzi; Zhang, Chi; and Tim Kraska, Mohammad Alizadeh; Papaemmanouil, Olga; Tatbul, Nesime: Neo: A Learned Query Optimizer. In: *PVLDB*. Jgg. 12, S. 1705–1718, 2019.
- [Ma21] Marcus, Ryan; Negi, Parimarjan; Mao, Hongzi; Tatbul, Nesime; Alizadeh, Mohammad; Kraska, Tim; Bao: Making Learned Query Optimization Practical. In: *SIGMOD*. S. 1275–1288, 2021.
- [MN06] Moerkotte, Guido; Neumann, Thomas: Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In: *VLDB*. S. 930–941, 2006.
- [MP18] Marcus, Ryan; Papaemmanouil, Olga: Deep Reinforcement Learning for Join Order Enumeration. In: *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. S. 1–4, 2018.
- [MP19] Marcus, Ryan; Papaemmanouil, Olga: Plan-Structured Deep Learning Models for Query Performance Prediction. *PVLDB*, 12(11):1733–1746, 2019.
- [Ne21] Negi, Parimarjan; Marcus, Ryan; Kipf, Andreas; Mao, Hongzi; Tatbul, Nesime; Kraska, Tim; Alizadeh, Mohammad: Flow-Loss: Learning Cardinality Estimates That Matter. *PVLDB*, 14(11):2019–2032, 2021.
- [PGH18] Pfannschmidt, Karlson; Gupta, Pritha; Hüllermeier, Eyke: Deep Architectures for Learning Context-dependent Ranking Functions. *arXiv preprint arXiv:1803.05796*, 2018.
- [SB13] Sasirekha, K.; Baby, P.: Agglomerative Hierarchical Clustering Algorithm - A Review. *International Journal of Scientific and Research Publications*, 3(3), 2013.
- [Ve21] Ventura, Francesco; Kaoudi, Zoi; Quiané-Ruiz, Jorge-Arnulfo; Markl, Volker: Expand your Training Limits! Generating Training Data for ML-based Data Management. In: *SIGMOD*. S. 1865–1878, 2021.
- [Wa18] Wang, Xuanhui; Li, Cheng; Golbandi, Nadav; Bendersky, Michael; Najork, Marc: The LambdaLoss Framework for Ranking Metric Optimization. In: *Proceedings of the 27th ACM international conference on information and knowledge management*. 2018.
- [Ya19] Yang, Zongheng; Liang, Eric; Kamsetty, Amog; Wu, Chenggang; Duan, Yan; Chen, Xi; Abbeel, Pieter; Hellerstein, Joseph M.; Krishnan, Sanjay; Stoica, Ion: Deep Unsupervised Cardinality Estimation. 13(3):279–292, 2019.
- [Yu20] Yu, Xiang; Li, Guoliang; Chai, Chengliang; Tang, Nan: Reinforcement Learning with Tree-LSTM for Join Order Selection. In: *ICDE*. S. 1297–1308, 2020.

# Pairwise Learning to Rank for Hit Song Prediction

Maximilian Mayerl,<sup>1</sup> Michael Vötter,<sup>2</sup> Günther Specht,<sup>3</sup> Eva Zangerle<sup>4</sup>

**Abstract:** Predicting the popularity of songs in advance is of great interest to the music industry, possible applications include assessing the potential of a new song, automated songwriting assistants, or song recommender systems. This task was traditionally solved by using pointwise models focused on single songs, either using classification to categorize songs into classes like *hit* and *non-hit*, or regression to predict popularity metrics like *play count*. In this work, we draw inspiration from research on learning to rank and instead use a pairwise model. Our model takes a pair of songs *A* and *B* and predicts whether song *A* is more popular than song *B*. We present a neural network model that is trained in a pairwise fashion, as well as two data augmentation strategies for improving its performance. We also compare our model to one trained in a traditional pointwise manner. Our experiments show that the pairwise model using our proposed augmentation strategies outperforms the pointwise model.

**Keywords:** Learning to Rank; Hit Song Prediction; Pairwise Ranking

## 1 Introduction

*Hit song prediction*, also called *song popularity prediction*, is a common task in music information retrieval. Its goal is to predict how popular a particular song is going to be, where popularity is usually defined in terms of sales, streaming counts, or a proxy like chart positions. Solving this task is of high interest to members of the music industry. A record label could use it to screen potential new releases, decide whether to release a record or to determine how much to invest in a particular record's promotion. Musicians could use it to get automated feedback while writing songs. Music retailers or streaming sites could employ it to augment their recommender systems.

In general, hit song prediction is often framed as a classification or regression problem. In the classification case, models are trained to divide songs into classes like *popular* and *unpopular*, while in the regression case, models predict popularity measures such as a song's top chart position or listening counts. As their input, hit song prediction models typically use intrinsic features of the song's audio signal, ranging from low-level features like Mel spectrograms to high-level features like danceability. Some models also incorporate song lyrics.

---

<sup>1</sup> Universität Innsbruck maximilian.mayerl@uibk.ac.at

<sup>2</sup> Universität Innsbruck michael.voetter@uibk.ac.at

<sup>3</sup> Universität Innsbruck guenther.specht@uibk.ac.at

<sup>4</sup> Universität Innsbruck eva.zangerle@uibk.ac.at

At its core, hit song prediction can also be viewed as a ranking problem—ordering songs by their popularity provides an implicit ranking of songs by popularity. This is also reflected in how song popularity is often communicated in the form of charts. To model hit song prediction for our approach, we borrow from a related field in information retrieval—learning to rank [Li11]. Most existing approaches to hit song prediction have one thing in common: they seek to predict the popularity—either in the form of a class label or a continuous popularity measure—of a *single* song; they are pointwise ranking models. In learning to rank (LTR), pointwise models have historically been superseded by pairwise and listwise models, which show better results than pointwise approaches. We, therefore, propose to tackle hit song prediction using a pairwise approach, solving the following task: Given a pair of songs  $A$  and  $B$ , predict whether song  $A$  is *more popular* than song  $B$ .

The core contributions of this paper are: (1) We propose a neural network model that takes the audio features of two songs as its input and outputs a label  $y \in \{0, 1\}$ , indicating whether the first song is more popular than the second song. (2) We propose two approaches to augment the data used for training this model, and show that both of these approaches improve the performance of the resulting model. (3) We compare the performance of our model against a model that is trained in the traditional, pointwise way—i.e., as a regressor predicting a popularity score for a single song—and show that the pairwise model performs better for ranking songs by popularity than the pointwise model. (4) For our experiments, we construct a dataset based on data from MusicBrainz<sup>5</sup>, AcousticBrainz<sup>6</sup>, and Last.fm<sup>7</sup>, which we make publicly available.

## 2 Related Work

Given its big potential impact on the music industry, a body of research has investigated possible solutions, with mixed results. Originally, there was doubt whether predicting song popularity based on audio features of a song was even possible. To that end, Pachet and Roy [PR08] performed a classification experiments using audio features, and concluded that “hit song science is not yet a science”. Later approaches were more successful, and used either classification or regression models. The machine learning models employed for this include support vector machines [LL18, DL05], boosting classifiers [DL05], shifting perceptrons [Ni11], and more complex neural network architectures [Za19, Ya17, Yu17]. What most of those approaches have in common is that they only consider single songs at a time; their models take individual songs, and seek to predict their popularity. However, song popularity does not exist in isolation. Songs are competing with each other for the attention and time of music listeners.

Therefore, our approach aims to take this competitive context into account by looking at pairs of songs. To the best of our knowledge, there exists only one similar approach. Yu et

---

<sup>5</sup> <https://musicbrainz.org>

<sup>6</sup> <https://acousticbrainz.org>

<sup>7</sup> <https://www.last.fm>

al. [Yu17] use a pair of CNNs with shared parameters, which they then train using a loss function incorporating a ranking loss. This makes their CNNs predict popularity scores that reflect the relative popularity ranking of the songs. However, different from our approach, their model still produces a scalar popularity score for every song, whereas our approach aims to directly answer whether one song is more popular than another.

### 3 Dataset

To conduct our experiments, we require a dataset of songs with both audio features as well as popularity information. Such a dataset needs to have the following properties: (1) It has to be large enough to make training of a neural network model with a large number of parameters feasible. (2) The songs have to follow a realistic distribution of popularity, to avoid introducing biases into models trained on the dataset. (3) The dataset needs to feature a popularity measure that can be used equally well for popular and unpopular songs. (4) Ideally, it should be publicly available or consist only of data that is publicly available, to make our research reproducible. To our knowledge, no such dataset is currently publicly available. Therefore, we propose a new dataset, which we also make available publicly for other researchers via Zenodo<sup>8</sup>. Datasets that were previously used for hit song prediction are either too small (with only a few thousand songs), have a biased (unrealistic) distribution of popularity (e.g., are balanced in hits and non-hits), use popularity measures that are not suitable for unpopular songs (e.g., chart positions only being available for popular songs), or are not publicly available. A comparison of our dataset to the ones used in prior research is given in Table 1. For the construction of our dataset, we used three sources of data: MusicBrainz<sup>9</sup> for song metadata, AcousticBrainz<sup>10</sup> for acoustic features, and Last.fm<sup>11</sup> for play counts as popularity measure. As can be seen from Table 1, our dataset is the largest, features a realistic popularity distribution, and uses a popularity measure that equally addresses popular and unpopular songs. In the following, we describe the creation of the dataset.

#### 3.1 Song Metadata

MusicBrainz is an openly available database of music metadata. It provides information about artists, recordings, releases, etc., and gets its data via crowd-sourcing. We use this database as our ground set of songs. Since MusicBrainz contains information about a large number of songs, both popular and unpopular, we reason that a random sample of songs drawn from its database should give us a set of songs that is reasonably representative in terms of the distribution of their popularity. We do note, however, that songs only known to

<sup>8</sup> <https://zenodo.org/record/7525833#.Y77d-7XMJaY>

<sup>9</sup> <https://musicbrainz.org>

<sup>10</sup> <https://acousticbrainz.org>

<sup>11</sup> <https://www.last.fm>

| Used By                   | Size     | PD  | PM  | PA  |
|---------------------------|----------|-----|-----|-----|
| Pachet and Roy [PR08]     | 32,000   | no  | yes | no  |
| Dhanaraj and Logan [DL05] | 1,700    | no  | no  | no  |
| Lee and Lee [LL18]        | 16,686   | no  | no  | no  |
| Lee and Lee [LL18]        | 1,264    | no  | no  | no  |
| Ni et al. [Ni11]          | 5,947    | no  | no  | no  |
| Zangerle et al. [Za19]    | 11,646   | no  | no  | yes |
| Yang et al. [Ya17]        | ~125,000 | no  | yes | no  |
| Our Dataset               | 142,963  | yes | yes | yes |

Tab. 1: A comparison of datasets used in prior research and our dataset. Size is given in number of songs. PD: Is the popularity distribution of songs in the dataset realistic? PM: Is the popularity measure used in the dataset suitable for popular as well as unpopular songs? PA: Is the dataset publicly available?

very few people are unlikely to have an entry in MusicBrainz, which could introduce a slight bias towards more popular music. For our dataset, we used a dump of the MusicBrainz database obtained at the beginning of 2020. From the songs in the database, we drew a random sample of approximately 20%, providing us with 3,407,667 songs.

### 3.2 Acoustic Features

After obtaining our ground set of songs, the next step was to gather audio data for these songs. For this, we used AcousticBrainz, a project in the same vein as MusicBrainz to collect a set of rich audio features for as many songs as possible. These audio features are obtained by contributors of AcousticBrainz running a client application, which performs feature extraction using the open-source Essentia library [Bo13b, Bo13a] and then uploads the results to the AcousticBrainz platform. We chose to use AcousticBrainz because (1) its data is indexed by MusicBrainz IDs, making it easy to merge the audio data with our set of songs, (2) it provides audio features for a large collection of songs, and (3) Essentia is a well-known feature extractor which is frequently used in music information retrieval.

For our dataset, we used the newest AcousticBrainz data dump, released in January 2015. This limits our dataset to songs that were released before that date. This way, we were able to obtain audio data for 201,047 of the 3,407,667 songs in our ground dataset. Via AcousticBrainz, we added the following audio features to our dataset, which are a subset of the features provided by AcousticBrainz<sup>12</sup> (we mainly retained features describing the whole song, as opposed to individual time slices of a song): 71 high-level audio features (mood and genre descriptors, gender of the singer, etc.), 236 low-level audio features (loudness, dissonance, spectral features like MFCCs, etc.), 57 rhythmic features (beats per minute, onset rate, etc.), and 45 tonal features (the key of the song, the tuning frequency, etc.).

<sup>12</sup> [https://essentia.upf.edu/streaming\\_extractor\\_music.html#music-descriptors](https://essentia.upf.edu/streaming_extractor_music.html#music-descriptors)

| Property           | Count   | Mean       | Std. Dev.  | Min  | Max          | Median  |
|--------------------|---------|------------|------------|------|--------------|---------|
| Number of Songs    | 142,963 | -          | -          | -    | -            | -       |
| Number of Artists  | 25,667  | -          | -          | -    | -            | -       |
| Number of Features | 409     | -          | -          | -    | -            | -       |
| Songs per Artist   | -       | 5.57       | 14.25      | 1.00 | 588.00       | 2.00    |
| Play Count         | -       | 100,046.28 | 464,534.19 | 0.00 | 16,668,020.0 | 4,325.0 |

Tab. 2: Summary of the properties of our dataset.

### 3.3 Popularity Measure

The last step for constructing our dataset was adding a suitable popularity measure. Along the lines of Schedl [Sc16], we used the play counts provided by Last.fm for this. Last.fm is a popular music listening platform that can be queried using MusicBrainz IDs, making it easy to crawl the play counts for the songs in our dataset. Using play counts as a popularity measure has the benefit of providing a natural measure for the popularity of all songs, as opposed to only for popular songs like chart position or similar measures. Of the 201,047 songs with audio features, we were able to obtain a play count for 146,075 songs. Finally, we removed songs for which audio features were missing (i.e., had no value), resulting in a total of 142,963 songs for our final dataset. Those songs were performed by 25,667 distinct artists. Table 2 features an overview of the dataset.

## 4 Methods

Our approach to hit song prediction relies on a pairwise model which, given a pair of songs  $A$  and  $B$ , attempts to predict whether song  $A$  is more popular than song  $B$ . This approach is grounded in research on learning to rank [Li11], where pairwise models have superseded pointwise approaches. We propose a neural network model that takes the audio features (cf. Section 3.2) of two songs as its input and produces a binary label answering the above question as output. Further, we present two techniques for augmenting the training data to further improve the model’s performance. In the following, we present the model, followed by our approaches toward training data augmentation.

### 4.1 Pairwise LTR Model

The model we propose is a quite straightforward neural network architecture. We employ a feed-forward network with six hidden layers of sizes between 256 and 64 neurons, and an output layer with a single neuron. A schematic depiction of our network architecture is given in Figure 1. The hidden layers all use SELU activation, as proposed by Klambauer et al. [Kl17]. We chose SELU units because they showed the best results in preliminary experiments, where we tested them against ELU, ReLU and tanh activations.

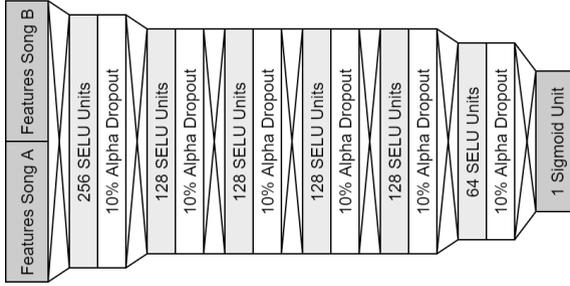


Fig. 1: A schematic depiction of our model.

As described by Klambauer et al. [Kl17], we initialized the weights of the network with values drawn from a normal distribution with zero mean and unit variance, as is necessary to obtain the self-normalizing property of SELU units. We also use the modified dropout variant proposed by Klambauer et al., with a dropout factor of 10%. For the output layer, we use a single neuron with sigmoid activation. The output is a value  $y' \in [0, 1]$ , which is then mapped to a label  $y \in \{0, 1\}$ , where  $y = 1$  signifies that song A is more popular than song B, and  $y = 0$  signifies that the opposite is the case. The mapping is done via

$$y = \begin{cases} 0 & \text{if } y' < 0.5 \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

## 4.2 Training Data Augmentation

To improve the classification performance of our model, we propose two augmentation strategies for the training data that aim to force the model to focus on those aspects that potentially make a song more popular than another. Note that these augmentation strategies are only possible due to the model being trained in a pairwise fashion—they could not be applied to a pointwise model. In the following, we explain those strategies and our reasoning behind them. Note that these augmentation strategies work independently of each other and always operate on the base dataset. In other words, samples generated by one augmentation step are not used as the input to another step.

### 4.2.1 Mirrored Training

The first augmentation strategy we propose is *mirrored training*. It works as follows: If our training dataset contains a pair of songs  $x = (A, B)$  with label  $y$ , we add the pair  $x' = (B, A)$  with label  $y'$  as defined in the following equation to the training set:

$$y' = \begin{cases} 0 & \text{if } y = 1 \\ 0 & \text{if } y = 0 \text{ and } A \text{ and } B \text{ have same play count} \\ 1 & \text{otherwise} \end{cases} \quad (2)$$

In other words, we mirror the pair of songs and invert the label (or leave it as 0, if both songs have the same play count, meaning that none is more popular than the other). We expect this augmentation strategy to help the model with learning to work independently of the order of the two input songs.

### 4.2.2 Reflexive Training

The second augmentation strategy we propose is *reflexive training*. It works as follows: If our training dataset contains a pair of songs  $x = (A, B)$ , regardless of the label, we add the following two instances to the training data:

- $(A, A)$ , with label  $y = 0$
- $(B, B)$ , with label  $y = 0$

In other words, we add pairs for both songs compared with themselves. The label is 0 in both cases since no song can be more popular than itself. Our reasoning for this strategy is that it should force the model to focus on *comparing* the two songs to determine which one is the more popular one, instead of focusing on the features of one of the songs. Without this augmentation, we expect that the model could, for example, simply learn to classify whether the first song in a pair is popular, essentially ignoring the second song.

## 5 Experiments

Given our model and dataset, we performed the following experiments to evaluate the performance of our pairwise model and the effectiveness of our augmentation strategies against a traditional pointwise model.

### 5.1 Generating Pairs of Songs

To obtain training and testing data for our pairwise model, we have to draw random pairs of songs from our dataset that we described in Section 3. For this, we first split our dataset

into training (64,175 songs), test (47,178 songs), and validation set (31,610 songs). The validation set was only used for preliminary experiments. For training and testing, we took the training and test sets, respectively, and then generated pairs of songs from each set as follows. Suppose we want to draw  $n$  pairs. We then generate two lists of length  $n$  of random integers in the range  $[1, m]$  with  $m = 142,963$  being the number of songs in the set. Those two lists are then used as the indices of the first and second song in the pairs, respectively. This means that if, for example, the two lists were  $[1212, 2342, 1, 42, \dots]$  and  $[96, 232, 3636, 5, \dots]$ , the first pair would consist of songs number 1212 and 96, the second pair of songs 2342 and 232, etc. For the labels, we use the Last.fm play count of song  $X$ ,  $\text{pc}(X)$ , and for pair  $(A, B)$  set  $y$  as follows:

$$y = \begin{cases} 1 & \text{if } \text{pc}(A) > \text{pc}(B) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

## 5.2 Experiments for the Pairwise Model

To verify the performance of our pairwise model and the effectiveness of our augmentation strategies, we performed a set of four experiments:

- **pairwise:** An experiment with our model as described in Section 4, without using any of our augmentation strategies.
- **pairwise-mirror:** An experiment with the proposed model, but using the *mirrored training* augmentation strategy.
- **pairwise-reflexive:** An experiment with the same model, but using the *reflexive training* augmentation strategy.
- **pairwise-full:** An experiment with the same model, and using both the *mirrored training* and *reflexive training* augmentation strategies.

For our experiments, we drew 300,000 pairs of random songs for the training and 100,000 pairs for the test set. The random seed was fixed, so that all experiments were performed with the same base set of song pairs. For testing, we also performed mirroring augmentation if the model was trained with mirroring. We never performed reflexive augmentation for the test set, as in a practical application the model would never be used to predict if a song is more popular than itself. For all experiments, our model was trained using binary cross-entropy loss using the Adam optimizer [KB14]; we trained for 100 epochs.

## 5.3 Comparison: Pointwise Model

To judge how well our pairwise model performs compared to traditionally trained pointwise models for hit song prediction, we also performed experiments with a model trained in a

| Model                     | Mirrored Training | Reflexive Training | F <sub>1</sub> | Precision    | Recall       |
|---------------------------|-------------------|--------------------|----------------|--------------|--------------|
| <b>pairwise-full</b>      | yes               | yes                | <b>0.670</b>   | 0.622        | <b>0.726</b> |
| <b>pairwise-mirror</b>    | yes               | no                 | 0.640          | 0.637        | 0.642        |
| <b>pairwise-reflexive</b> | no                | yes                | 0.626          | <b>0.653</b> | 0.602        |
| <b>pairwise</b>           | no                | no                 | 0.617          | 0.633        | 0.602        |
| <b>pointwise</b>          | n/a               | n/a                | 0.629          | 0.627        | 0.631        |

Tab. 3: The results of our experiments.

traditional way—i.e., as a pointwise regression model predicting the play count of a given song. For this, we used the same general model architecture as for our pairwise model, to make results as comparable as possible, but reduced the input layer to only contain the features for a single song, as opposed to a pair of songs. For this experiment, the output layer activation function was removed, since we need the network to be able to output arbitrary values for predicting the play count of a given song. For training and testing, we used the same training and test set splits as outlined in Section 5.1. The training was done over 100 epochs, using mean squared error loss. For the evaluation, we randomly drew 100,000 pairs of songs from the testing set, using the same procedure as described in Section 5.1, and then used the model to predict the play counts for both songs in the pair. If the predicted play count of the first song was higher than that of the second song, we set the predicted label to 1, meaning that the model predicted the first song to be more popular, and otherwise we set it to 0. This procedure allows us to evaluate how well a pointwise model performs relative ranking—i.e., detecting which of two songs should be the higher ranked one—and makes it possible to compare the results of the pointwise and the pairwise model.

## 6 Results and Discussion

For the evaluation, we use the F<sub>1</sub> score, precision and recall metrics, since those metrics are widely used to measure the performance of classification models and provide a fair evaluation even if a dataset is not balanced, which should make our results more easily comparable to others. The results of the evaluation are given in Table 3. Looking at the pairwise models, a clear difference can be seen between the models using augmentation and the one which does not. The model without any form of data augmentation achieves an F<sub>1</sub> score of 0.617, which is outperformed by all models using augmentation. The models using only a single augmentation strategy show slight to moderate improvements, with an F<sub>1</sub> score of 0.626 for the model using reflexive training, and 0.640 for the model using mirrored training. Notably, mirrored training seems to primarily boost recall, improving from 0.602 for the model without augmentation to 0.642, while reflexive training seems to primarily boost precision, improving from 0.633 for the model without augmentation to 0.653. Combining both augmentation strategies leads to the best performance, with an F<sub>1</sub> score of 0.670. This shows that the proposed augmentation methods are individually effective and complement each other well.

The pointwise model achieves an  $F_1$  score of 0.629. Comparing to the pairwise models, we see that while the pointwise model perform better than the pairwise model trained without any augmentation, the models using a single augmentation strategy perform about as good as the regression models. The model combining both augmentation strategies clearly outperforms the regression models, with an  $F_1$  score of 0.670 compared to 0.629. This shows that training a model in a pairwise fashion and leveraging suitable augmentation strategies—which only becomes possible due to the model being a pairwise model—leads to a model that is better at predicting the relative popularity of two songs, at least in our setup.

We do note, however, that there are limitations to our results. First, while we tried to keep the bias towards more popular songs in our dataset as low as possible, we acknowledge that our data sources most probably still exhibit such a bias. Nonetheless, the popularity in our dataset shows a characteristic long tail distribution, implying that this bias is small. Second, our dataset is also biased towards slightly older songs, containing no songs released after January 2015. We note that, in the context of using play counts as a popularity measure, this, in fact, helps reduce a popularity bias in the data, as older songs naturally had more time to accrue plays, which would unfairly disadvantage newer songs.

## 7 Conclusion

In this paper, we proposed a pairwise approach for hit song prediction. We created a model which, given a pair of songs  $A$  and  $B$ , predicts whether song  $A$  is more popular than song  $B$ . For this, we proposed a neural network architecture which takes as its input audio features of two songs and thus can be trained to directly answer that question, as opposed to traditional pointwise approaches for popularity prediction. Furthermore, we proposed two data augmentation strategies to improve the performance of our model. Our results showed that both augmentation strategies are effective in improving the performance of our pairwise model, and that using both together leads to the best results. We also performed a comparison with a model trained in a traditional way—i.e., as a pointwise model—and showed that our pairwise model trained using both of our augmentation strategies clearly outperforms it.

Future work includes analyzing the importance of individual features, to determine which features allow a pairwise model to determine which song is more popular. Another possible direction for future work is to investigate whether other popularity measures (distinct listener count, chart positions, etc.) can also effectively be used with a pairwise model. As our results suggest that approaches from learning to rank seem to be transferable to hit song prediction, a natural next step would also be to use more sophisticated pairwise learning to rank models to predict the popularity of songs.

## Bibliography

- [Bo13a] Bogdanov, Dmitry; Wack, Nicolas; Gómez, Emilia; Gulati, Sankalp; Herrera, Perfecto; Mayor, Oscar; Roma, Gerard; Salamon, Justin; Zapata, José; Serra, Xavier: ESSENTIA: An Open-Source Library for Sound and Music Analysis. In: Proc. of the 21st ACM International Conference on Multimedia. pp. 855–858, 2013.
- [Bo13b] Bogdanov, Dmitry; Wack, Nicolas; Gómez Gutiérrez, Emilia; Gulati, Sankalp; Herrera Boyer, Perfecto; Mayor, Oscar; Roma Trepas, Gerard; Salamon, Justin; Zapata González, José Ricardo; Serra, Xavier: Essentia: An Audio Analysis Library for Music Information Retrieval. In: Proc. of the International Society for Music Information Retrieval Conference. pp. 493–498, 2013.
- [DL05] Dhanaraj, Ruth; Logan, Beth: Automatic Prediction of Hit Songs. In: Proc. of the International Society for Music Information Retrieval Conference. pp. 488–491, 2005.
- [KB14] Kingma, Diederik P; Ba, Jimmy: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
- [Kl17] Klambauer, Günter; Unterthiner, Thomas; Mayr, Andreas; Hochreiter, Sepp: Self-normalizing neural networks. In: Proc. of the 31st International Conference on Neural Information Processing Systems. pp. 972–981, 2017.
- [Li11] Liu, Tie-Yan: Learning to Rank for Information Retrieval. Springer Science & Business Media, 2011.
- [LL18] Lee, Junghyuk; Lee, Jong-Seok: Music Popularity: Metrics, Characteristics, and Audio-Based Prediction. *IEEE Transactions on Multimedia*, 20(11):3173–3182, 2018.
- [Ni11] Ni, Yizhao; Santos-Rodriguez, Raul; Mcvicar, Matt; De Bie, Tjil: Hit Song Science Once Again a Science. In: 4th International Workshop on Machine Learning and Music. 2011.
- [PR08] Pachet, François; Roy, Pierre: Hit Song Science Is Not Yet a Science. In: Proc. of the International Society for Music Information Retrieval Conference. pp. 355–360, 2008.
- [Sc16] Schedl, Markus: The lfm-1b dataset for music retrieval and recommendation. In: Proc. of the 2016 ACM on International Conference on Multimedia Retrieval. pp. 103–110, 2016.
- [Ya17] Yang, Li-Chia; Chou, Szu-Yu; Liu, Jen-Yu; Yang, Yi-Hsuan; Chen, Yi-An: Revisiting the Problem of Audio-based Hit Song Prediction Using Convolutional Neural Networks. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 621–625, 2017.
- [Yu17] Yu, Lang-Chi; Yang, Yi-Hsuan; Hung, Yun-Ning; Chen, Yi-An: Hit song prediction for pop music by siamese CNN with ranking loss. arXiv preprint arXiv:1710.10814, 2017.
- [Za19] Zangerle, Eva; Vötter, Michael; Huber, Ramona; Yang, Yi-Hsuan: Hit Song Prediction: Leveraging Low- and High-Level Audio Features. In: Proc. of the International Society for Music Information Retrieval Conference. 2019.



# Communication-Optimal Parallel Reservoir Sampling

Christian Winter,<sup>1</sup> Moritz Sichert,<sup>2</sup> Altan Birler,<sup>3</sup> Thomas Neumann,<sup>4</sup> Alfons Kemper<sup>5</sup>

**Abstract:** When evaluating complex analytical queries on high-velocity data streams, many systems cannot run those queries on all elements of a stream. Sampling is a widely used method to reduce the system load by replacing the input with a representative yet manageable subset. For unbounded data, reservoir sampling generates a fixed-size uniform sample independent of the input cardinality. However, the collection of reservoir samples itself can already be a bottleneck for high-velocity data.

In this paper, we introduce a technique that allows fully parallelizing reservoir sampling for many-core architectures. Our approach relies on the efficient combination of thread-local samples taken over chunks of the input without necessitating communication during the sampling phase and with minimal communication when merging. We show how our efficient merge guarantees uniform random samples while allowing data to be distributed over worker threads arbitrarily. Our analysis of this approach within the Umbra database system demonstrates linear scaling along the available threads and the ability to sustain high-velocity workloads.

**Keywords:** Reservoir Sampling; Parallel Sampling; Stream Processing

## 1 Introduction

With the widespread deployment of cheap connected sensors and the increased use of off-premise systems, there is an ever-growing need to analyze the log and data streams generated by these sensors and systems remotely. Due to this data's high volume and high velocity, analyzing all generated entries and values is often infeasible. Therefore, many analyses are performed on a reduced version of the data. Some applications rely on aggregates over windows or subsets of the data, e.g., monitoring average temperature readings in a given area. Others, e.g., for analyzing log streams, apply highly selective filters to reduce the input cardinality, showing only relevant error messages and surrounding entries. However, for some applications, representative and unfiltered data points are desirable. Sampling is employed to reduce the data stream to a manageable size for analytics systems.

By drawing a uniform sample from a stream of data points, each point of the stream has an equal probability of being part of the resulting sample, and thus the result is representative of the entire stream. Sampling is utilized in a wide range of applications, e.g., for workload

---

<sup>1</sup> Technische Universität München winterch@in.tum.de

<sup>2</sup> Technische Universität München moritz.sichert@tum.de

<sup>3</sup> Technische Universität München altan.birler@tum.de

<sup>4</sup> Technische Universität München neumann@in.tum.de

<sup>5</sup> Technische Universität München kemper@in.tum.de

statistics [BRN20, LN90], machine learning [Sc22, Sc21, Sc15], and big data [Ma20]. While some sampling algorithms take a variable-sized subset of the input, e.g., selecting  $x\%$  of the arriving tuples at random, their resulting sample size can still be infeasible for analyzing high-velocity unbounded data streams. Therefore, we will focus on those algorithms that produce a fixed-sized sample independent of the input size, i.e., reservoir sampling.

While reservoir sampling produces a uniform random sample in a single pass over the input in  $O(n(1 + \log(N/n)))$  [Li94], the sheer volume of data can lead to bottlenecks during the sampling phase. In the past, the growth in data could be compensated by the performance improvements of new hardware. However, with Moore's law coming to its end, a single core can often no longer keep up. Therefore, many solutions for stream processing and analysis, such as dedicated stream processing engines [Za16, Ze20, Ca15] and in-database stream-processing approaches [Wi20], have focused on processing incoming streams in a parallel and distributed manner.

In this paper, we describe a mechanism allowing fully parallel reservoir sampling without communication between sampling threads. By keeping thread-local samples over chunks of the input, we can construct a complete sample of size  $n$  in parallel from  $p$  worker samples using only  $2p + n$  messages in an efficient  $k$ -way merge. The low message overhead is especially beneficial in distributed environments, where communication takes place using comparably slow and expensive network connections. Further, we describe an  $O(p + n \log(p))$  merge strategy optimized for small sample sizes and show that it can guarantee uniform random samples, independent of how input elements are assigned to workers. By constraining all communication to our merge stage, our approach can scale almost linearly in many-core machines. Implementing our approach within the code-generating Umbra database system [NF20], we demonstrate that our approach is applicable in real-world systems. Using this implementation, we evaluate our novel merge strategy against a merge strategy based on a hypergeometric distribution in many-core applications for different sample sizes, showing that our proposed merge is beneficial for small sample sizes. Overall, our communication optimal reservoir sampling can scale linearly along the number of available workers and sustain a sampling throughput of more than 300 million tuples per second per thread, independent of the distribution used in the merge.

The remainder of the paper is structured as follows: We introduce relevant concepts and algorithms to our approach in Sect. 2 and discuss related work. In Sect. 3, we present the design and implementation of both of our contributions and prove the correctness of our novel merge strategy. To demonstrate our approach's applicability and performance, we evaluate its scalability and throughput in Sect. 4 before concluding in Sect. 5.

## 2 Background and Related Work

Approximate results are often deemed acceptable to gain instant query response times for analytics over large volumes of data. Simple random samples of fixed size are a reliable tool to reduce the costs of computing approximate statistics [SA22]. We will argue why this tool is particularly interesting and discuss related work in constructing such samples.

When using a random sample, a small random subset of the data is picked and processed instead of the entire data, significantly improving query response times. Other statistical tools, such as histograms [Co12] and distinct count sketches [FN19], are useful for approximate statistics. However, they are inflexible as the filters that they can evaluate are limited. A histogram can only evaluate simple predicates, such as one (or few) dimensional ranges. Samples, on the other hand, can evaluate arbitrary predicates, as they are smaller representatives of the entire data set. In the absence of complex filters, histograms can provide useful upper bounds, while samples can only provide probabilistic estimates. However, with large enough samples, the variance of the estimates a sample provides is relatively low and can be relied upon. Additionally, with complex filters, a histogram's upper bounds can be too high to be useful, as it can only consider simple range predicates.

There are many ways to pick a random sample. For computing unbiased statistics, simple random samples without replacement are a good fit as every subset of the data is selected with equal probability. Tillé [Ti11] describes the theoretical background of simple random sampling and computation of statistics from a simple random sample. Ting [Ti21] provides efficient implementations for a range of algorithms for sampling without replacement. We focus on samples of fixed size. In contrast to Bernoulli sampling, where every tuple is picked with independent probability  $\theta$ , fixed-size samples do not grow alongside the input data size. One might assume a larger sample would be a better fit for a larger data set. This is true for predicates whose selectivity decrease with increasing data set size, such as filters for fixed timespan on a data set that grows over time. However, for predicates of constant selectivity, the size of the data set has little to no effect on the quality of the sample. We will try to build a simplified intuition as to why and refer the reader to the detailed theoretical analysis by Moerkotte and Hertzschuch [MH20] for further details.

Given a sample of size  $n$ , a data set of size  $N = \rho n$ , and a predicate of constant selectivity  $\sigma$ , we want to estimate the number of matches  $K = \sigma N = \sigma n \rho$  where  $\rho$  is the ratio of the size of the data set to the size of the sample. This task is common in cardinality estimation within database systems [He21]. As a strategy, we evaluate the predicate on the sample and count the number of matches  $k$ , which is a random variable from the hypergeometric distribution  $HG(\rho n, \sigma \rho n, n)$ . We use the simple estimator  $\hat{K} = k\rho$  as our estimate of  $K = \sigma n \rho$ . As a simple cost metric, we define the expected relative mean squared error of our estimate as:

$$\text{rMSE} = \mathbb{E} \left[ \left( \frac{\hat{K} - K}{K} \right)^2 \right] = \mathbb{E} \left[ \left( \frac{k\rho - \sigma n \rho}{\sigma n \rho} \right)^2 \right] = \frac{1}{\sigma n} \text{Var} [k] = \frac{1 - \sigma}{\sigma} \frac{\rho - 1}{\rho n - 1} \approx \frac{1 - \sigma}{\sigma} \frac{1}{n} \quad (1)$$

Assuming  $\rho$  is relatively large, it disappears from our error estimate, implying that the relative size of the sample has little to no effect on the accuracy of this estimate. On the contrary, the predicate's selectivity and the sample's absolute size directly influence the error. An intuitive explanation of this result is that we can assume that the data set from which we are sampling is itself a random sample drawn from an infinite distribution. Resampling from this *intermediate sample* simply means that we construct a smaller sample of the original data set. The size of the intermediate sample has only a small effect on the final sample's contents. So, given requirements on the error and information on the selectivity of predicates, it makes sense to pick a *fixed* sample size rather than having sample sizes adapt to the data set size as adapting the size of a sample is a costly operation requiring that the sample be rebuilt.

The optimal way to compute a simple random sample depends on various factors. Our proposed approach focuses on concurrently sampling from many parallel data streams in environments where communication costs are high (we are optimal in the number of communications), and memory usage is not a limiting factor. Due to these assumptions, our approach is flexible and an excellent fit for distributed environments. There are simpler algorithms for when the size of the data set is known beforehand: The draw-by-draw procedure iterates over the sample and picks tuples one by one with potentially high costs from random accesses into the data [Ti11]. The sequential selection-rejection method described by Fan et al. [FMR62] instead iterates over the data to produce the sample. Vitter [Vi84, Vi87] further improves the selection rejection method by computing skip lengths; rather than iterating over the data one tuple at a time, one can probabilistically generate the number of tuples to skip before the next tuple is selected, significantly reducing the number of random number generations. This approach is parallelized by Sanders et al. [Sa16] by distributing the task of random sampling among different workers. Chickering et al. [CRM07] describe merging parallel reservoir samples using a hypergeometric distribution, which we utilize for larger sample sizes, and offer proof that the resulting sample is still uniformly random. However, they send all local samples to a centralized coordinator for the merge, leading to communication overhead.

To maintain a sample of size  $n$  in a single pass over a data stream, a set of more than  $n$  values, the so-called reservoir, needs to be processed to guarantee a simple random sample at any point during processing. All procedures maintaining a simple random sample in a single pass over a data stream are variants of the reservoir sampling algorithm defined by Vitter [Vi85]. Reservoir sampling iterates over input tuples and selects them for the sample with a probability proportional to the number of tuples seen so far. Vitter [Vi85] improves on this by probabilistically generating skips, a contiguous amount of tuples not contained in the sample, thus reducing the costs for generating random numbers from  $\mathcal{O}(N)$  to  $\mathcal{O}(n(1 + \log(N/n)))$ . Li [Li94] improves on the approach by Vitter by proposing a simpler distribution to generate skips. These sequential approaches only support sampling data from a single stream. Hübschle-Schneider and Sanders [HS20] also parallelize reservoir sampling by independently collecting multiple reservoirs and merging them afterward. However,

their approach needs to maintain a distributed priority queue, which incurs additional communication costs but potentially reduces the sizes of their independent samples. For situations where memory is scarce, Tirthapura and Woodruff [TW11] maintain a single sample at a central coordinator. Their approach has optimal communication complexity for the centralized sample setting. Birler et al. [BRN20] reduce the communication costs of Tirthapura and Woodruff’s approach by accepting temporary imperfections in the central sample that are eventually corrected. Our approach, in contrast, is communication optimal among all possible distributed reservoir sampling algorithms. For this property, we accept a slight increase in per-stream memory consumption, which is acceptable in analytical workloads as our local samples are fixed-sized and small compared to all the other data the streams need to maintain.

The performance characteristics of the various approaches can be analyzed by looking at communication costs, total processing costs, and total memory use. These three metrics are influenced by the utilized sampling approach, the sample size, the data size, and the number of workers. Shared-sample-based approaches [BRN20, TW11] benefit from low memory use and total processing costs. Thus, they are well applicable to low communication cost environments, such as a single machine with a single CPU socket. However, in settings with high communication costs, such as manycore machines with multiple sockets or distributed networks, distributed sampling approaches [HS20] are beneficial as they sacrifice memory and some computation to cut down on inter-worker communication. These trade-offs are necessary as Tirthapura and Woodruff [TW11] prove that communication costs may not be optimal when only one shared sample exists. In our approach, where we focus on minimizing communication costs, we must maintain a full sample per worker. Otherwise, we must incur additional communication or provide weaker guarantees, such as a probability of failure [Sa16].

### 3 Approach

Having outlined the background in sampling, we can describe our merge-based parallel reservoir sampling technique and the novel post-sampling merge strategy optimized for small sample sizes. Our approach aims to draw a sample of  $n$  tuples uniformly random in parallel from an input of  $N$  tuples using  $p$  workers. Each worker  $i$  draws a thread-local reservoir sample of size  $n$  out of the  $N_i$  tuples assigned to it, denoted as  $s_{i,1} \dots s_{i,n}$ , using algorithm  $L$  [Li94]. This sample is merged into a global sample on demand. We assume no prior knowledge about the workload, only the reservoir size  $n$  has to be known, and we materialize only tuples selected for a local reservoir. Throughout this section, we will assume a work-stealing, morsel-based [Le14] distribution of input chunks to workers, as this is the parallelization strategy of our system Umbra. However, our approach is applicable to any input distribution strategy. Our contributions are twofold. First, we detail the communication-optimal merge process, which improves upon prior work independent of the reservoir size and merge strategy used. Subsequently, we discuss our novel merge strategy optimized for small reservoir sizes.

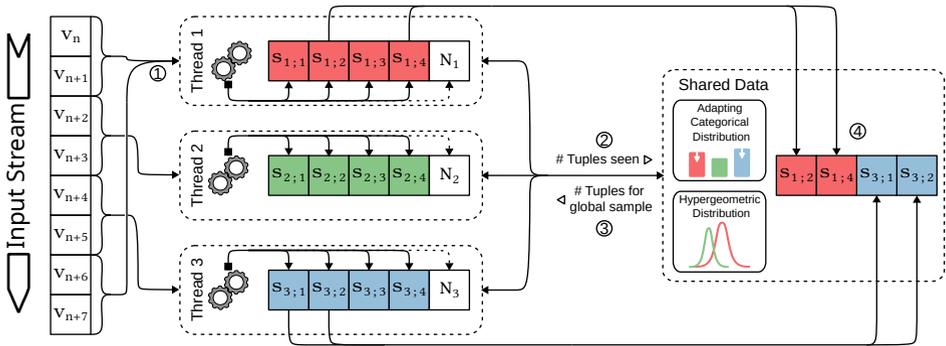


Fig. 1: Overview of the sampling process with three workers consisting of ① thread-local sampling, ② the transfer of local cardinalities, ③ determining thread shares in the global sample, and ④ the transfer of the resulting sample tuples.

### 3.1 Communication-Optimal Process

Our approach consists of two phases, the sampling and the merge phase, as shown in Fig. 1. In the sampling phase ①, all threads create local reservoirs of size  $n$  over chunks of the input. While sampling, workers fetch arriving chunks independently from one another, ensuring that each chunk is assigned to exactly one worker. Reservoir sampling guarantees that all local samples are uniformly random at any point, with each tuple having a probability of  $\frac{n}{N_i}$  to be contained in the corresponding thread-local sample. To generate the whole sample, each worker first reports the cardinality of all chunks it processed,  $N_i$ , to a coordinator ②.

This coordinator can be an external worker or one of the sampling workers. First, the coordinator determines the share in tuples that each thread-local sample has in the global sample using either the hypergeometric distribution, or the proposed merge strategy outlined in detail below. Then, the coordinator notifies each thread of the number of tuples it has to choose for the global sample ③, which in turn selects the desired amount uniformly at random from their sample and reports it to the global sample ④. In contrast to prior work [CRM07] sharing all local samples with the coordinator in  $pn$  messages, our approach needs at most  $2p + n$  messages: 2 per worker to communicate the local cardinality and the number of tuples to contribute to the global sample, and  $n$  to send the selected tuples.

### 3.2 Merge Strategy for Small Reservoirs

Conceptually, our approach for small reservoirs relies on iteratively evaluating and updating a categorical distribution over all threads for each position of the final sample to determine which thread to select for this reservoir slot. In contrast to strategies based on the hypergeometric distribution, the categorical distribution has to be evaluated per sample slot and

**Algorithm 1** Calculating per-thread share of global sample

---

```

1: function CALCULATETHREADSHARE(localCardinalities[])
2:   globalCardinality  $\leftarrow$  sum(localCardinalities)
3:   fenwickTree  $\leftarrow$  FenwickTree::build(localCardinalities)
4:   samplesPerThread  $\leftarrow$  []
5:   slot  $\leftarrow$  0
6:   while slot < sampleSize and globalCardinality > 0 do
7:     selectedTuple  $\leftarrow$  pickRandom(0, globalCardinality - 1)
8:     selectedThread  $\leftarrow$  fenwickTree.rank(selectedTuple)
9:     samplesPerThread[selectedThread]  $\leftarrow$  samplesPerThread[selectedThread] + 1
10:    fenwickTree.add(selectedTuple, -1)
11:    globalCardinality  $\leftarrow$  globalCardinality - 1
12:    slot  $\leftarrow$  slot + 1
13:  return samplesPerThread

```

---

not per thread. While this is too costly for large sample sizes, it avoids the computationally expensive hypergeometric distribution. Each thread  $i$  is selected with a probability of  $\frac{N_i}{N}$  for the first slot. As we sample without replacement, we decrease the cardinality of the selected thread  $N_i$  and the global cardinality  $N$  by 1 for the next draw. All threads  $j$  with  $j \neq i$  have the probability  $\frac{N_j}{N-1}$  of being selected for the next reservoir slot, the selected thread has a probability of  $\frac{N_i-1}{N-1}$ . We repeat this until we have selected the source for all  $n$  spaces in the sample, decreasing  $N$  and  $N_i$  for the selected  $i$  at every draw. Note that while conceptually drawing slot by slot, we do not care about the order of the sample or the actual slot to select from. This allows us to only track the number of tuples per thread.

Algorithm 1 shows our implementation. In the first step, we calculate the global cardinality from the thread-local information and build a Fenwick tree over the local cardinalities (Line 3). Fenwick trees, as described in [Fe94], allow efficient operations over prefix sums while requiring only linear space. Building a Fenwick tree is possible in linear time, whereas rank and update operations have logarithmic runtime. Following the setup, we can pick the shares for each thread. For this, we first generate a random number  $r$  in the range of 0 to  $N$  (Line 7). For this, we first generate a random integer  $r \in [0, N)$  (Line 7). From this value, we pick the corresponding thread by using the prefix-sums stored in the Fenwick tree. For  $r \in [0, N_1)$ , we pick thread 1, for  $r \in [N_1, N_2)$ , thread 2, and so forth. Mapping  $r$  to a thread this way is possible in  $\mathcal{O}(\log(p))$  using the rank operation of the Fenwick tree (Line 8). Then, we update the cardinality of the selected thread and the global cardinality (Lines 10 and 11) for the next draw from the updated categorical distribution. We repeat the draw and distribution update until we have either selected every tuple or filled every slot in the final sample.

Our algorithm does not require floating point arithmetic, allowing a fast and exact evaluation. The Fenwick tree construction dominates the setup step with a runtime of  $\mathcal{O}(p)$ . The loop of Line 6 is evaluated  $n$  times, requiring  $\mathcal{O}(\log(p))$  to update the Fenwick tree, resulting in an overall runtime complexity of  $\mathcal{O}(p + n \log(p))$ .

### 3.3 Proof

To show that the merge strategy outlined above does not change the resulting samples' probability, we show that it selects tuples from local reservoirs equal to the hypergeometric distribution. For this, we will use the probability mass function  $P(X = k)$  where  $X$  denotes the number of tuples selected from thread  $i$ . For our proof, we use the fact that the positions for which  $i$  is selected are irrelevant. Consider first the case where all  $k$  selections of  $i$  happen in the first  $k$  draws, followed by  $n - k$  draws of  $j \neq i$ . This results in the probability

$$P(\text{first } k \text{ from } i) = \frac{N_i}{N} \times \frac{N_i - 1}{N - 1} \times \cdots \times \frac{N_i - k + 1}{N - k + 1} \times \frac{N - N_i}{N - k} \times \frac{N - N_i - 1}{N - k - 1} \times \cdots \times \frac{N - N_i - n + k + 1}{N - n + 1} \quad (2)$$

$$= \frac{N_i \times (N_i - 1) \times \cdots \times (N_i - k + 1) \times (N - N_i) \times (N - N_i - 1) \times \cdots \times (N - N_i - n + k + 1)}{N \times (N - 1) \times \cdots \times (N - n + 1)}. \quad (3)$$

It is clear that, through the commutative property of the product, the draws for  $i$ ,  $N_i$  to  $N_i - k + 1$ , can be moved to any of the draws  $N$  to  $N - n + 1$  without changing the resulting probability. For the full  $P(X = k)$ , we additionally need to select the  $k$  positions for our  $i$  draws, resulting in the probability

$$P(X = k) = \frac{N_i \times (N_i - 1) \times \cdots \times (N_i - k + 1) \times (N - N_i) \times (N - N_i - 1) \times \cdots \times (N - N_i - n + k + 1)}{N \times (N - 1) \times \cdots \times (N - n + 1)} \times \binom{n}{k}. \quad (4)$$

Using  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$  and  $x \times (x - 1) \times \cdots \times (x - m + 1) = \frac{x!}{(x-m)!}$  we get

$$P(X = k) = \frac{(N_i \times (N_i - 1) \times \cdots \times (N_i - k + 1)) \times ((N - N_i) \times (N - N_i - 1) \times \cdots \times (N - N_i - n + k + 1))}{N \times (N - 1) \times \cdots \times (N - n + 1)} \times \binom{n}{k} \quad (5)$$

$$= \frac{N_i!}{(N_i - k)!} \times \frac{(N - N_i) \times (N - N_i - 1) \times \cdots \times (N - N_i - n + k + 1)}{N \times (N - 1) \times \cdots \times (N - n + 1)} \times \binom{n}{k} \quad (6)$$

$$= \frac{N_i!}{(N_i - k)!} \times \frac{(N - N_i)!}{(N - N_i - n + k)!} \times \frac{1}{N \times (N - 1) \times \cdots \times (N - n + 1)} \times \binom{n}{k} \quad (7)$$

$$= \frac{N_i!}{(N_i - k)!} \times \frac{(N - N_i)!}{(N - N_i - n + k)!} \times \frac{(N - n)!}{N!} \times \frac{n!}{k!(n - k)!} \quad (8)$$

$$= \frac{N_i!}{k!(N_i - k)!} \times \frac{(N - N_i)!}{(N - N_i - (n - k))! \times (n - k)!} \times \frac{(N - n)!n!}{N!} \quad (9)$$

$$= \frac{\binom{N_i}{k} \times \binom{N - N_i}{n - k}}{\binom{N}{n}} \quad (10)$$

which is the probability mass function of the hypergeometric distribution.

## 4 Evaluation

Having outlined the implementation of our fully parallel communication-optimal reservoir sampling approach, we demonstrate its performance, focussing on scalability and the impact of our proposed merge strategy for small sample sizes. The experiments are conducted using an implementation within Umbra on a server equipped with 2 AMD EPYC™ 7713 CPUs with 64 cores each and 1 TiB of main memory. To reduce the impact of IO bottlenecks, we generate all data using the PostgreSQL-derived `generate_series`<sup>6</sup> command. All results reported are based on averages over 9 runs, each sampling  $N = 100$  billion records.

<sup>6</sup> <https://www.postgresql.org/docs/current/functions-srf.html>

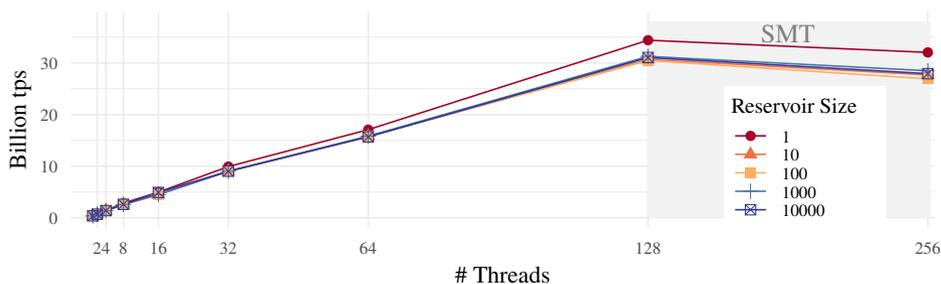


Fig. 2: Total sampling throughput with an increasing number of threads: Our communication-optimal sampling approach scales nearly linearly to more than 30 billion tuples per second.

#### 4.1 Scalability and Performance

In the first experiment, we investigate the scalability of our approach. As we require no communication between threads during the sampling phase, we expect the sampling phase to scale perfectly along the thread count. Additionally, the runtime of the merge phase is independent of the input size, so we expect it to amortize for large data sets. We measure the total throughput of our implementation with different reservoir sizes and an increasing number of threads and report the results in Fig. 2. As expected, the throughput of processed tuples scales nearly linearly with the number of threads. For a reservoir size of 1, our implementation can process up to 35 billion tuples per second when using all 128 physical cores. The experiment samples 8 B integers, so in total, our system processes up to 280 GB of data per second.

Our approach requires each thread to collect a full-size local sample. For this reason, the memory usage of sampling increases linearly with the number of threads. Therefore, we expect higher sampling overhead and lower throughput for increasing sample sizes. However, the results show that the overhead is manageable even for larger sample sizes. For example, even when collecting a sample of size 10000, our implementation can still process over 30 billion tuples per second.

#### 4.2 Merge Strategy Comparison

Our approach requires communication between threads only in the merge phase. We want to show that our merge strategy, which employs a categorical distribution, can be more efficient than a hypergeometric distribution while producing equivalent results. To evaluate our strategy, we compare the runtime of the merge phase for both distributions. Fig. 3 shows the relative speedup of our merge strategy with varying numbers of threads and sample sizes. Note that the merge phase itself always runs single-threaded on a coordinator node.

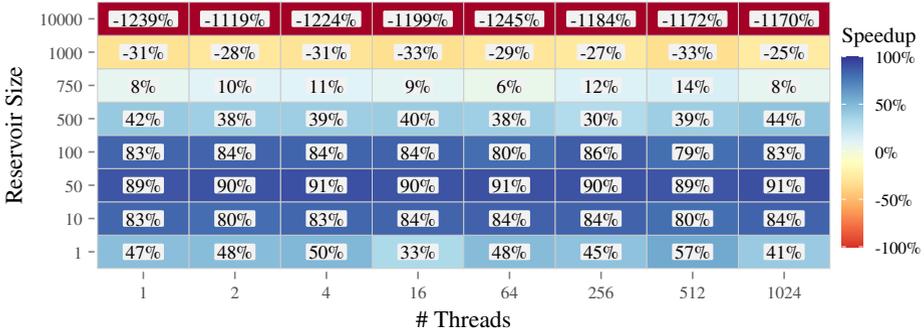


Fig. 3: Speedup of our merge strategy based on a categorical distribution over using a hypergeometric distribution: For sample sizes below 1000, our approach achieves up to 91% speedup independent of the number of threads.

The number of threads in the figure refers to the number of locally collected samples to be merged.

For sample sizes up to 750, our approach consistently outperforms using the hypergeometric-distribution-based merge. The main reason for the efficiency of our merge strategy is that it uses no floating-point operations. However, we need to perform two operations on the Fenwick tree for every element in the sample, while the merge phase using a hypergeometric distribution only generates one value from the distribution for every thread, independent of the sample size. Therefore, our merge strategy is inefficient for larger sample sizes. The experiment further shows that our approach’s speedup is generally independent of the number of threads: Our strategy consistently achieves similar speedup across all sample sizes, even for several hundred threads.

## 5 Conclusion

In this paper, we introduced a new communication-optimal parallel reservoir sampling technique, requiring only  $2p + n$  messages for environments with  $p$  workers and a sample size of  $n$ . Our technique relies on an efficient merge of thread-local reservoir samples, each taken over arbitrarily distributed chunks of the input. In addition, we described a novel merge strategy optimized for small sample sizes and provide proof that this strategy is statistically equal to the hypergeometric distribution. With our implementation of communication-optimal parallel reservoir sampling in the Umbra database system, we evaluated its overall performance, and both merge strategies. Achieving more than 370 million samples per thread, we showed near-linear scale up to 128 workers and a clear advantage of our optimized merge for samples smaller than 1000 tuples.

## Bibliography

- [BRN20] Birler, Altan; Radke, Bernhard; Neumann, Thomas: Concurrent online sampling for all, for free. In: DaMoN. ACM, pp. 5:1–5:8, 2020.
- [Ca15] Carbone, Paris; Katsifodimos, Asterios; Ewen, Stephan; Markl, Volker; Haridi, Seif; Tzoumas, Kostas: Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.
- [Co12] Cormode, Graham; Garofalakis, Minos N.; Haas, Peter J.; Jermaine, Chris: Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends Databases*, 4(1-3):1–294, 2012.
- [CRM07] Chickering, David M; Roy, Ashis K; Meek, Christopher A.: Distributed reservoir sampling for web applications, December 11 2007. US Patent 7,308,447.
- [Fe94] Fenwick, Peter M.: A New Data Structure for Cumulative Frequency Tables. *Softw. Pract. Exp.*, 24(3):327–336, 1994.
- [FMR62] Fan, C. T.; Muller, Mervin E.; Rezucha, Ivan: Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital Computers. *Journal of the American Statistical Association*, 57(298):387–402, 1962.
- [FN19] Freitag, Michael J.; Neumann, Thomas: Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2019.
- [He21] Hertzschuch, Axel; Moerkotte, Guido; Lehner, Wolfgang; May, Norman; Wolf, Florian; Fricke, Lars: Small Selectivities Matter: Lifting the Burden of Empty Samples. In: SIGMOD Conference. ACM, pp. 697–709, 2021.
- [HS20] Hübschle-Schneider, Lorenz; Sanders, Peter: Communication-Efficient Weighted Reservoir Sampling from Fully Distributed Data Streams. In: SPAA. ACM, pp. 543–545, 2020.
- [Le14] Leis, Viktor; Boncz, Peter A.; Kemper, Alfons; Neumann, Thomas: Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In: SIGMOD Conference. ACM, pp. 743–754, 2014.
- [Li94] Li, Kim-Hung: Reservoir-Sampling Algorithms of Time Complexity  $O(n(1 + \log(N/n)))$ . *ACM Trans. Math. Softw.*, 20(4):481–493, 1994.
- [LN90] Lipton, Richard J.; Naughton, Jeffrey F.: Query Size Estimation by Adaptive Sampling. In: PODS. ACM Press, pp. 40–46, 1990.
- [Ma20] Mahmud, Mohammad Sultan; Huang, Joshua Zhexue; Salloum, Salman; Emara, Tamer Z.; Sadatdiynov, Kuanishbay: A survey of data partitioning and sampling methods to support big data analysis. *Big Data Min. Anal.*, 3(2):85–101, 2020.
- [MH20] Moerkotte, Guido; Hertzschuch, Axel: alpha to omega: the G(r)eek Alphabet of Sampling. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2020.
- [NF20] Neumann, Thomas; Freitag, Michael J.: Umbra: A Disk-Based System with In-Memory Performance. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2020.
- [Sa16] Sanders, Peter; Lamm, Sebastian; Hübschle-Schneider, Lorenz; Schrade, Emanuel; Dachs-bacher, Carsten: Efficient Random Sampling - Parallel, Vectorized, Cache-Efficient, and Online. *CoRR*, abs/1610.05141, 2016.

- [SA22] Sanca, Viktor; Ailamaki, Anastasia: Sampling-Based AQP in Modern Analytical Engines. In: DaMoN. ACM, pp. 4:1–4:8, 2022.
- [Sc15] Schelter, Sebastian; Soto, Juan; Markl, Volker; Burdick, Douglas; Reinwald, Berthold; Evfimievski, Alexandre V.: Efficient sample generation for scalable meta learning. In: ICDE. IEEE Computer Society, pp. 1191–1202, 2015.
- [Sc21] Schüle, Maximilian E.; Lang, Harald; Springer, Maximilian; Kemper, Alfons; Neumann, Thomas; Günemann, Stephan: In-Database Machine Learning with SQL on GPUs. In: SSDBM. ACM, pp. 25–36, 2021.
- [Sc22] Schüle, Maximilian E.; Lang, Harald; Springer, Maximilian; Kemper, Alfons; Neumann, Thomas; Günemann, Stephan: Recursive SQL and GPU-support for in-database machine learning. *Distributed Parallel Databases*, 40(2):205–259, 2022.
- [Ti11] Tillé, Yves: Sampling Algorithms. In: *International Encyclopedia of Statistical Science*, pp. 1273–1274. Springer, 2011.
- [Ti21] Ting, Daniel: Simple, Optimal Algorithms for Random Sampling Without Replacement. CoRR, abs/2104.05091, 2021.
- [TW11] Tirthapura, Srikanta; Woodruff, David P.: Optimal Random Sampling from Distributed Streams Revisited. In: DISC. volume 6950 of *Lecture Notes in Computer Science*. Springer, pp. 283–297, 2011.
- [Vi84] Vitter, Jeffrey Scott: Faster Methods for Random Sampling. *Commun. ACM*, 27(7):703–718, 1984.
- [Vi85] Vitter, Jeffrey Scott: Random Sampling with a Reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.
- [Vi87] Vitter, Jeffrey Scott: An efficient algorithm for sequential random sampling. *ACM Trans. Math. Softw.*, 13(1):58–67, 1987.
- [Wi20] Winter, Christian; Schmidt, Tobias; Neumann, Thomas; Kemper, Alfons: Meet Me Halfway: Split Maintenance of Continuous Views. *Proc. VLDB Endow.*, 13(11):2620–2633, 2020.
- [Za16] Zaharia, Matei; Xin, Reynold S.; Wendell, Patrick; Das, Tathagata; Armbrust, Michael; Dave, Ankur; Meng, Xiangrui; Rosen, Josh; Venkataraman, Shivaram; Franklin, Michael J.; Ghods, Ali; Gonzalez, Joseph; Shenker, Scott; Stoica, Ion: Apache Spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.
- [Ze20] Zeuch, Steffen; Chaudhary, Ankit; Monte, Bonaventura Del; Gavriilidis, Haralampos; Giouroukis, Dimitrios; Grulich, Philipp M.; Breß, Sebastian; Traub, Jonas; Markl, Volker: The NebulaStream Platform for Data and Application Management in the Internet of Things. In: CIDR. [www.cidrdb.org](http://www.cidrdb.org), 2020.

# CLOCQ: A Toolkit for Fast and Easy Access to Knowledge Bases

Philipp Christmann,<sup>1</sup> Rishiraj Saha Roy,<sup>2</sup> Gerhard Weikum<sup>3</sup>

**Abstract:** Curated knowledge bases (KBs) store vast amounts of factual world knowledge, and are therefore ubiquitous in many information retrieval (IR) and natural language processing (NLP) applications like question answering, named entity disambiguation, or knowledge exploration. Despite that, accessing information from complete knowledge bases is often a daunting task. Researchers and practitioners typically have crisp use cases in mind, for which standard querying interfaces can be overly complex and inefficient. We aim to bridge this gap, and release a public toolkit that provides functionalities for common KB access use cases, and make it available via a public API. Experiments show efficiency improvements over existing KB interfaces for various important functionalities.

**Keywords:** Knowledge Base; Knowledge Graph; RDF; Efficiency

## 1 Introduction

Large curated knowledge bases (KBs), also known as knowledge graphs (KGs), like Wikidata [VK14], DBpedia [Au07], YAGO [SKW07], Freebase [Bo08], and industrial counterparts (e.g. at Amazon, Apple, Google, or Microsoft), store *factual* world knowledge in compact RDF (Resource Description Framework) triples.

Such knowledge bases empower question answering (QA) systems [BH15, Be13, Ch19, SRA22], that offer natural interfaces for accessing information. Most digital personal assistants like Alexa, Siri, Google Assistant, or Cortana are essentially QA systems accessing a variety of information at their backends, including curated KBs. There is a wide range of research on QA, ranging from methods that target simple single-shot questions [Ab18, Be13], spanning methods dedicated for complex multi-hop questions [Su18], to algorithms that keep track of an ongoing conversational context [CSRW22b, KSRW21, LJ21, Sa18].

For implementing QA systems, one can identify a common set of basic KB functionalities that are very often necessary. For example, retrieving *all KB facts with a specific entity*, or computing the shortest KG path between two entities, are two such frequent needs.

Apart from QA, there is a range of other tasks/applications that benefit from quick and easy access to large KBs. Named entity recognition and detection (NERD) systems can be

---

<sup>1</sup> Max-Planck-Institut für Informatik & Saarland University, Saarbrücken, Deutschland, pchristm@mpi-inf.mpg.de

<sup>2</sup> Max-Planck-Institut für Informatik & Saarland University, Saarbrücken, Deutschland, rishiraj@mpi-inf.mpg.de

<sup>3</sup> Max-Planck-Institut für Informatik & Saarland University, Saarbrücken, Deutschland, weikum@mpi-inf.mpg.de

used for mapping entities in text to canonicalized objects, which can give insights about a text at hand [FS10, Ho11, Li20]. Another example use case is entity ranking [CD22], which features in search engines to show entity-centric information for queries like “*Angela Merkel*”. Further, information or statistics extracted from the KB can be leveraged for improving performance on downstream tasks: the distance between entities can serve as a proxy for their semantic similarity [ZI16], the frequency of an entity in the KB can be used for understanding how popular an entity is [Ch19], and the KB ontology can help in relation extraction [Ko14] or answer verification [BH15].

For implementing such methods, several shared KB functionalities are required: identifying the distance between two entities, retrieving the frequency of a KB item, retrieving the type of an entity, or computing the shortest path between two KB items.

Available interfaces to large KBs, with multiple terabytes of data, are often based on query languages like SPARQL. Such interfaces allow for a very general access to the KB with arbitrary complexity, and are heavily optimized for different query patterns and workloads [Fe13, Ur16]. However, implementing some of the basic operations mentioned above can lead to a high degree of query complexity, manual effort and efficiency overhead.

One key problem is that using existing interfaces for accessing KBs *requires deep knowledge and understanding of the respective KB schema*, which is different for every KB. Another problem that we identify is that the whole *KB storage and the corresponding query languages are optimized for the native RDF triple structure*. However, modern KBs store *n*-ary facts, using *reification* (e.g. via qualifier statements in Wikidata, or Compound Value Types (CVTs) in Freebase), going beyond the self-contained triples [HHK15]. In fact, one fourth of the facts in Wikidata provide additional information via such qualifier statements. Consider the real-life fact that Angela Merkel was chancellor of Germany, in triple structure:

```
⟨Angela Merkel, position held, Federal Chancellor of Germany⟩
```

From this fact, important contextual information is missing: she was chancellor in the past, from 2005 to 2021, she was the 8th German chancellor, she replaced Gerhard Schröder, etc. In triple structure, this additional context is represented by reification of the basic fact, and adding more triples that refer to the basic fact’s identifier and express the contextual information. The single *n*-ary (compound) fact discussed above would be represented as:

```
⟨Angela Merkel, position held, fact-id⟩
⟨fact-id, position held, Federal Chancellor of Germany⟩
  ⟨fact-id, start time, 2005⟩
  ⟨fact-id, end time, 2021⟩
  ⟨fact-id, series ordinal, 8⟩
  ⟨fact-id, replaces, Gerhard Schröder⟩
```

Using traditional query interfaces, retrieving *all facts* with Angela Merkel from the knowledge base can be cumbersome and inefficient: queries with the entity as subject, object, and

qualifier-object of the fact are required. An example SPARQL query for detecting all facts with Angela Merkel as subject from Wikidata is shown below:

```
SELECT DISTINCT ?fact_id ?subject ?predicate ?object ?qual_pred ?qual_obj {
  VALUES (?subject) {(Angela Merkel)}
  ?subject ?p ?fact_id .
  ?fact_id ?ps ?object .
  ?predicate wikibase:claim ?p .
  ?predicate wikibase:statementProperty ?ps .
  OPTIONAL{
    ?fact_id ?pq ?qual_obj .
    ?qual_pred wikibase:qualifier ?pq
  }
}
```

Similar queries need to be run binding Angela Merkel to the object and qualifier-object position. Afterwards, the results need to be post-processed, joining all constituents for one fact-id. Overall, this causes quite a few KB interactions and processing overhead.

Further, certain KB concepts are not well-defined for such scenarios. How should this fact be represented in graphical form, where graphs of all facts would be overlaid to obtain a knowledge graph? Would 2009 belong to the 1-hop neighborhood of Barack Obama? What would be the KB distance between Barack Obama and George W. Bush? Should the shortest path include non-content like fact-ids?

We present a fact-centric definition of the KB, which represents KB facts as arbitrary-length lists, considering context information in qualifiers to be of comparable importance as that in the main triple<sup>4</sup>. This allows us to establish intuitive definitions for KB neighborhood, KB distance, and shortest paths between entities. Based on these definitions, we implement a fact-centric KB index, CLOCQKB, that allows efficient KB access to perform the functionalities described above. The resulting KB interface is made available to the community for accessing KBs more conveniently, both as a public API and open-source code repository.

**Contributions.** Our contributions in this work are as follows:

- Identifying and outline key problems with existing interfaces for common KB access needs in many NLP and IR use cases;
- Proposing an efficient solution to the problems with existing triple-centric interfaces, based on a fact-centric view of the KB;
- Proposing concise and unambiguous definitions for basic KB concepts;
- Making our code available, and release a public API that allows people to conveniently access Wikidata, without having to deal with multiple terabytes of data<sup>5</sup>.

<sup>4</sup> This implements and extends our work in WSDM 2022 [CSRW22a].

<sup>5</sup> <https://clocq.mpi-inf.mpg.de>

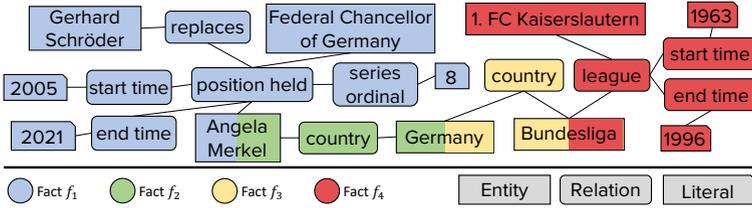


Fig. 1: Graph representation of a KB fact.

## 2 KB Index

Our primary design choice is to take a *fact-centric* view of the KB. Therefore, we treat the KB as a set of *facts*, instead of a set of triples. As a result, we also directly store and index the facts as a whole. With that, we avoid any complex query structures or post-hoc joining of separate fact constituents at runtime. This helps us to significantly improve the runtime efficiency of several important KB functionalities, like retrieving all KB facts with a specific KB item, or computing the distance between two KB items.

### 2.1 Concepts and definitions

Based on a fact-centric view of the KB, we establish the following definitions, answering the questions posed in Sec. 1.

**Knowledge base.** A *knowledge base*  $K$  is a set of KB facts.

**KB fact.** A *KB fact*  $f$  is a list of KB items, and expresses objective information or knowledge. It consists of the main triple, which has a subject, predicate, and object, and an optional set of qualifiers represented as (qualifier-predicate, qualifier-object) pairs. The subject is an entity, the predicate and qualifier-predicate are relations. The object is an entity, type or literal (date or string), and the qualifier-object can be an entity or literal. An example fact is  $\langle \text{Angela Merkel}, \text{position held}, \text{Federal Chancellor of Germany}; \text{start time}, 2005; \text{end time}, 2021; \text{series ordinal}, 8; \text{replaces}, \text{Gerhard Schröder} \rangle$ , which represents the same information as the compound fact in Sec. 1. A set of facts can be represented in a graph as illustrated in Fig. 1. In this representation, the qualifiers are connected to the predicate of a fact, describing the relation between the subject and object in more detail. Note also, that each instance of a relation becomes an individual node. If, for example, the two *country* nodes were merged, information would be lost:  $\langle \text{Bundesliga}, \text{country}, \text{Angela Merkel} \rangle$  could be inferred as a fact then.

**KB item.** A *KB item*  $x$  is either an entity (*Angela Merkel*), relation (*position held*), type (human), or literal (2021).

**Neighborhood.** The 1-hop neighborhood  $N_f$  of an item  $x$  is the set of *all facts* in  $K$  with  $x$ :  $N_f(x) = \{f | x \in f \wedge f \in K\}$ . To generalize, the  $h$ -hop neighborhood  $N_f^h(x)$  is then the union of all facts with any of the items in the  $(h-1)$ -hop neighborhood.

**Neighbors.** The 1-hop neighbors  $N_i$  of  $x$  are all KB items in the 1-hop neighborhood of  $x$ :  $N_i(x) = \{x' | x' \in f \wedge f \in N_f(x)\}$ . Analogously, the set of  $h$ -hop neighbors  $N_i^h(x)$  is given by all KB items in the  $h$ -hop neighborhood.

**Frequency.** The frequency of a KB item is given by the size of its 1-hop neighborhood  $|N_f(x)|$ , i.e. the number of facts  $x$  appears in. Without loss of generality, the frequency can be measured w.r.t. a specific position in the fact (e.g. frequency in subject-position).

**KB distance.** The KB distance between two KB items  $x$  and  $y$  is  $h$ , if the items are  $h$  hops away from each other, i.e.  $h = \min_{h'} x \in N_i^{h'}(y)$ . For example, the distance of Angela Merkel and Gerhard Schröder is 1, since they appear in the same fact. The distance between Angela Merkel and Bundesliga is 2.

**Shortest path.** The shortest path between two KB items is given by the (set of) fact(s) in between. This ensures that important contextual information is not skipped, considering the facts as a whole. For example, if we used only the direct connection between Angela Merkel and Federal Chancellor of Germany in the graph, incorrect inferences could be made. Instead, the whole fact  $f_1$  is the shortest path. The shortest path between Angela Merkel and Bundesliga would be  $f_2 \circ f_3$ , where  $\circ$  denotes concatenation.

## 2.2 Implementation

For improving space efficiency, each KB item  $x$  is first integer-encoded as  $int(x)$  [Fe13, Ur16], and we create mappings from  $x \rightarrow int(x)$  and  $int(x) \rightarrow x$ . Each fact  $f = \langle x_0, x_1, \dots \rangle$  is then stored as  $int(f) = \langle int(x_0), int(x_1), \dots \rangle$ . For facilitating our computations, we index the following information for each item in the KB: the 1-hop neighborhood  $N_f(x)$  and the set of 1-hop neighbors  $N_i(x)$ , both in an integer-encoded manner. This allows us to compute the most important functionalities via simple lookups or set operations at runtime, improving efficiency. Details on the specific implementations of individual functionalities can be found in Sec. 3.

The neighborhoods for each item  $x$  are stored in two lists, holding pointers to the individual facts: one list for facts with  $x$  in subject-position, and one for all other facts with  $x$ . This will be useful later, for retrieving only salient facts with a KB item. The neighborhood lists of  $x$  are then stored in position  $int(x)$  in the index, which is implemented by another list.

The neighbors of an item  $x$  are stored as a set, which is again stored in position  $int(x)$  in the corresponding index list. Another possibility would be to extract the set of neighbors from the 1-hop neighborhood at runtime. This reduces the memory footprint, but increases the costs for computing KB distances.

### 3 Functionalities

In this section, we will describe the functionalities provided with our KB interface in more detail, elaborating some of the implementation details. All functions are implemented for Wikidata, the largest public KB that is actively maintained. However, the principles would still hold for other large curated KBs like DBpedia as well.

#### 3.1 Direct lookups

**Label.** To avoid collisions due to duplicate labels, KBs typically use identifiers for representing KB items. For example, *Angela Merkel* is stored as `Q567` in Wikidata, and `position held` as `P39` (for simplicity, we use labels to refer to items in this paper). We provide a simple function to look up the (English) label for a provided Wikidata ID.

**Aliases.** Another useful information on a KB item are aliases. These are alternative labels that one can use for the same KB item. For example, “*CR7*” is an alias of *Cristiano Ronaldo*, and “*office held*” is an alias of `position held`. Such aliases can be used for improving relation extraction [Ba21, Va18] or NERD systems [BOM15]. Relation aliases can also be used for training crisp paraphrase models.

**Description.** Further, Wikidata stores a crisp description for each KB item, which can e.g. be helpful for deriving latent representations of an item [GSR17].

**Types.** Another important information on an entity are the KB types. Typical use cases are entity linking [GSR17], answer verification [BH15] in QA systems, or enhanced efficiency [Zi17] of QA systems. For example, *Angela Merkel* is a *human*, and *Germany* is associated with the types *sovereign state*, *republic*, and *country*. To enhance the expressiveness of the types, we add the occupations of a human, which are not stored as types in Wikidata. E.g. *Angela Merkel* would also be associated with *politician* or *physicist*.

**Most frequent type.** In some use cases, having exactly one type for an entity is desirable [CSRW22b]. Unlike the deprecated KB Freebase, Wikidata does not indicate the most notable type of an entity. As a proxy, we use the most frequent type of an entity in the KB.

#### 3.2 More complex functionalities

**1-hop neighborhood.** This function is used for retrieving the 1-hop neighborhood of an item, which is a frequent use case in QA [Ch19, SRA22, Su18], but can also be useful for other applications like KB completion [Ba19] or entity alignment [Su20]. For implementing this function, we can simply look up the neighborhood in our fact-centric KB index. We further implement a mechanism to retrieve the more salient facts for a specific KB item: frequent KB items like *Germany* can easily have millions of facts in their neighborhood. However, retrieving all these facts is often not desired in IR or NLP applications [SRA22].

A parameter  $p$  is used to control the amount of facts returned as follows: if there are more than  $p$  facts with  $x$  in the object or qualifier-object position, then these facts are dropped, i.e. only facts with  $x$  in the subject-position are kept. This can also help improve the efficiency of downstream applications: the I/O time can be drastically reduced, and only a subset of more salient facts needs to be processed. The output of this function is a set of KB facts.

**Frequency.** For computing the frequency, we count the number of facts with  $x$  as subject, and the number of facts with  $x$  not in subject position, using the neighborhood index. The output of this function are the two resulting counts.

**Connectivity.** We can also compute a connectivity score for two KB items. Note that when retrieving the 3 or 4-hop neighborhood of an item, this will give almost the entire KB. Therefore, we define a connectivity score of two KB items, which we found more useful than a standard distance function in practice [CSRW22a]. The connectivity is 1 if the items have a KB distance of 1, 0.5 if they have a distance of 2, and 0 if they are not connected within 2 hops. The function can be efficiently implemented using basic set operations:

$$connectivity(x, y) = \begin{cases} 1 & \text{if } x \in N_i(y) \vee y \in N_i(x) \\ 0.5 & \text{if } N_i(x) \cap N_i(y) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Note that the 1-hop neighbors  $N_i(x)$  and  $N_i(y)$  are simply looked up in the index.

**Shortest path.** Another often-needed KB functionality is obtaining the shortest path between two items, which can be used to analyze their semantic relationship [JJ19], obtaining training data for QA [Su18], or for connecting multiple subgraphs [Pr21]. The implementation makes use of the connectivity function outlined above: if two items  $x, y$  have a connectivity of 1, we search the smaller of the two 1-hop neighborhoods for a fact with both items. For a connectivity of 0.5, we identify the “items in the middle”  $\{m_i\}$ , i.e. the set of items that are connected with  $x$  in 1 hop and with  $y$  in 1 hop. For these items  $\{m_i\}$ , we search for the joint facts with  $x$  and  $y$ , respectively. The output of this function is a set of facts (connectivity = 0.5), or a set of 2-hop paths, one for each  $m_i$ , where each such path is given by a set of facts connecting  $x$  and  $m_i$ , and a set of facts connecting  $m_i$  and  $y$  (connectivity = 1).

### 3.3 Search Space Reduction

Finally, we briefly introduce the key algorithm introduced in our recent work [CSRW22a], which retrieves a set of relevant KB facts for a given question. This is typically the first step in QA systems [CSRW22b, SRA22, Su18] for reducing the search space. Consider an input like “*Who scored the final goal for Germany in 2014?*”. For a human interested in football it is obvious that “*Germany*” refers to the German national football team, and that the question is on the 2014 FIFA World Cup. However, this only becomes clear when considering the question as a whole, taking the respective context into account. Therefore, mapping the question words to KB items is a non-trivial task.

We propose a method that first retrieves a set of candidate disambiguations for each question word. These candidate disambiguations are then scored based on four different signals. We use the word-level scores for term matching and question relatedness. Further, to understand the text as a whole, we also consider the connectivity and coherence among disambiguations. Note that certain question words may be more ambiguous than others, and providing additional candidate disambiguations for such question words could alleviate potential errors. Therefore, we dynamically adjust the number of disambiguations  $k$  for each individual question word. The algorithm provides the disambiguations, and facts with the disambiguated items, as the output.

**Entity and relation linking.** Based on this algorithm, we also make functionalities for entity linking and relation linking available. The idea is to prune less relevant entities or relations from the disambiguations provided by the search space reduction method outlined above. These functions have been developed using data from the SMART 2022 Task<sup>6</sup>.

## 4 Experiments

In this section, we conduct experiments to compare the efficiency of our fact-centric KB interface with existing triple-centric ones on a set of important KB functionalities.

### 4.1 Experimental Setup

The CLOCQKB is initiated with a Wikidata dump from 31 January 2022, that has been cleaned for removing information irrelevant for most common tasks<sup>7</sup>. We show runtimes of a locally running version of CLOCQKB, a version without the neighbors' index (CLOCQKB w/o  $N_i$ ), and the publicly available CLOCQKB-API<sup>8</sup>.

We compare our fact-centric KB interface with two triple-centric ones: HDT [Fe13] which implements fast triple lookups based on efficient bitmap encodings, and the publicly available QUERYSERVICE<sup>9</sup> provided by Wikidata. For HDT, we used the latest dump made available (3 March 2021)<sup>10</sup>, and run it on our local machine. Note that it is highly non-trivial to perform the cleaning steps (mentioned above) for the HDT dump, since it comes in a highly compressed form. While function results may therefore not be exactly the same, the runtimes are still comparable. All experiments with the QUERYSERVICE are conducted in September 2022. Note that we measure all network latencies (if any), and subtract these from the measured timings for fair comparisons. The local experiments are run on an AMD EPYC 7702 Processor, and the public API is running on an AMD EPYC 7302P Processor.

---

<sup>6</sup> <https://smart-task.github.io/2022/>

<sup>7</sup> <https://github.com/PhilippChr/wikidata-core-for-QA>

<sup>8</sup> <https://clocq.mpi-inf.mpg.de>

<sup>9</sup> <https://query.wikidata.org>

<sup>10</sup> <https://www.rdfhdt.org/datasets/>

| KB interface  | HDT [Fe13]              | CLOCQKB                 | CLOCQKB w/o $N_i$       | QUERYSERVICE | CLOCQKB-API             |
|---------------|-------------------------|-------------------------|-------------------------|--------------|-------------------------|
| RAM consumed  | 150 GB                  | 470 GB                  | 360 GB                  | –            | –                       |
| Neighborhood  | 1.21 s                  | $5.99 \times 10^{-5}$ s | $6.01 \times 10^{-5}$ s | 0.561 s      | $4.36 \times 10^{-3}$ s |
| Frequency     | $3.12 \times 10^{-2}$ s | $1.02 \times 10^{-5}$ s | $1.15 \times 10^{-5}$ s | 0.122 s      | $4.24 \times 10^{-3}$ s |
| Connectivity  | 0.802 s                 | $1.83 \times 10^{-5}$ s | $5.02 \times 10^{-5}$ s | 1.11 s       | $4.22 \times 10^{-3}$ s |
| Shortest path | 3,046 s                 | 0.553 s                 | 15.8 s                  | 1.18 s       | 0.591 s                 |

Tab. 1: Large-scale runtime efficiency analysis of KB interfaces.

## 4.2 Large-scale efficiency analysis

We first compare the efficiency of the KB interfaces for the following four functionalities: i) retrieving the 1-hop **neighborhood**, ii) computing the **frequency**, iii) computing the **connectivity**, and iv) identifying the **shortest path**. We restrict our experiments on the more complex functionalities (Sec. 3.2) here. However, even for accessing the simpler functionalities (Sec. 3.1), deep knowledge of the KB schema is required for the baselines.

As input, we use 10,000 random item (pairs when applicable) for each functionality, using the same random seed for all interfaces. We only used 100 random pairs for shortest path, due to substantially higher runtimes. In case an error is thrown for a function call, which might sometimes occur for the QUERYSERVICE, the corresponding instance is dropped from the analysis for all methods. The results are shown in Table 1.

The key observations are as follows. Due to its smart KB indices, CLOCQKB can achieve extremely low runtimes for retrieving 1-hop neighborhoods, frequencies, or performing connectivity checks. Removing the neighbors' index only has a minor effect on the connectivity check, while the effect on the shortest path functionality is notable. The runtime of the triple-centric baselines is higher by a factor of  $10^3$  to  $10^5$  than that of CLOCQKB in most scenarios, indicating the runtime benefits of a fact-centric approach for such use cases. The public CLOCQKB-API has higher runtimes than the version running locally, due to a different processor and some I/O overhead, but is still substantially faster than the baselines.

Note that CLOCQKB consumes quite some RAM. One could store the indexes in a dedicated database to save memory. However, in an industrial use case, ~500 GB RAM is not a major concern, and for scientific use cases the CLOCQKB-API can be used.

## 4.3 Anecdotal examples

Results in Table 1 are for a random sample, and can show trends among the different KB interfaces. We also conducted experiments on a set of popular items that may often appear in a real application. We chose the following KB items: Angela Merkel, Germany, Bundesliga, and run the same functionalities on these items. Results can be seen in Table 2. Time-outs are indicated by "n/a".

| KB interface                             | HDT [Fe13]              | CLOCQKB                 | QUERYSERVICE            | CLOCQKB-API             |
|--|-------------------------|-------------------------|-------------------------|-------------------------|
| Neighborhood(Angela Merkel)              | 20.8 s                  | $2.55 \times 10^{-3}$ s | 2.12 s                  | $1.07 \times 10^{-2}$ s |
| Neighborhood(Germany)                    | 2,990 s                 | 2.73 s                  | "n/a"                   | 15.6 s                  |
| Neighborhood(Bundesliga)                 | 15.2 s                  | $1.10 \times 10^{-2}$ s | "n/a"                   | $3.56 \times 10^{-2}$ s |
| Frequency(Angela Merkel)                 | $2.85 \times 10^{-2}$ s | $2.55 \times 10^{-5}$ s | 0.186 s                 | $5.34 \times 10^{-3}$ s |
| Frequency(Germany)                       | $5.20 \times 10^{-5}$ s | $2.56 \times 10^{-5}$ s | 0.280 s                 | $5.39 \times 10^{-3}$ s |
| Frequency(Bundesliga)                    | $5.20 \times 10^{-5}$ s | $2.47 \times 10^{-5}$ s | $8.33 \times 10^{-2}$ s | $5.44 \times 10^{-3}$ s |
| Connectivity(Angela Merkel, Germany)     | 61.3 s                  | $3.48 \times 10^{-5}$ s | "n/a"                   | $5.37 \times 10^{-3}$ s |
| Connectivity(Germany, Bundesliga)        | 60.3 s                  | $3.27 \times 10^{-5}$ s | "n/a"                   | $5.21 \times 10^{-3}$ s |
| Connectivity(Angela Merkel, Bundesliga)  | 0.328 s                 | $8.28 \times 10^{-4}$ s | "n/a"                   | $5.10 \times 10^{-3}$ s |
| Shortest path(Angela Merkel, Germany)    | 118 s                   | $7.80 \times 10^{-2}$ s | "n/a"                   | $8.42 \times 10^{-2}$ s |
| Shortest path(Germany, Bundesliga)       | 120 s                   | $8.10 \times 10^{-2}$ s | "n/a"                   | $8.89 \times 10^{-2}$ s |
| Shortest path(Angela Merkel, Bundesliga) | 5,260 s                 | 0.156 s                 | "n/a"                   | 0.178 s                 |

Tab. 2: Runtime efficiency analysis on manually chosen function calls for popular entities.

The results reveal that the public `QUERYSERVICE` cannot cope with use cases in which larger intermediate results are obtained (like for Germany or even Bundesliga). Most such runs faced server-side time-outs. `HDT` obtains results for all experiments, but can take quite some time to do so: runtimes  $> 1$  s are rarely acceptable in IR or NLP applications. While runtimes for `CLOCQKB` can get higher in extreme cases, like for retrieving the 1-hop neighborhood of Germany with more than 1.4 million facts, the runtimes remain tractable, indicating runtime benefits of  $10^3$  to  $10^5$  of our fact-centric approach in most cases.

## 5 Related Work

There has been substantial work on optimizing query performance on KBs [EM10, NW08, Ur16]. Jacobs [UJ20] proposed `TRIDENT`, for enabling different kinds of workloads (e.g. SPARQL, graph analytics) on large KBs. `HDT` [Fe13] is an efficient representation of RDF data, that is both space and runtime efficient. The individual triples of the KB are encoded using bitmaps. Two integer-streams holding all predicates and objects for a fixed subject are established. The relations between these predicates and objects are encoded using bit-streams. Using several indexes, `HDT` can search for triple pattern very efficiently. However, there is no dedicated mechanism for querying  $n$ -ary facts.

While the proposed approaches show high performance on traditional SPARQL(-like) queries, we showed through comparison that important KB functionalities can be more efficiently implemented taking a fact-centric view of the KB.

## 6 Conclusion

We present a fact-centric view of the KB, and based on this, provide simple and efficient implementations of several important KB functionalities. These are either not available in existing implementations, or would be quite complex to use, especially for new users. Experiments show that we outperform existing KB interfaces w.r.t. runtime efficiency. We make our code available, and provide a public API to enhance KB applications and research.

## Bibliography

- [Ab18] Abujabal, Abdalghani; Saha Roy, Rishiraj; Yahya, Mohamed; Weikum, Gerhard: Never-ending learning for open-domain question answering over knowledge bases. In: WWW. pp. 1053–1062, 2018.
- [Au07] Auer, Sören; Bizer, Christian; Kobilarov, Georgi; Lehmann, Jens; Cyganiak, Richard; Ives, Zachary: DBpedia: A nucleus for a Web of open data. In: The Semantic Web. pp. 722–735, 2007.
- [Ba19] Bansal, Trapit; Juan, Da-Cheng; Ravi, Sujith; McCallum, Andrew: A2N: Attending to neighbors for knowledge graph inference. In: ACL. pp. 4387–4392, 2019.
- [Ba21] Bastos, Anson; Nadgeri, Abhishek; Singh, Kuldeep; Mulang, Isaiah Onando; Shekar-pour, Saeedeh; Hoffart, Johannes; Kaul, Manohar: RECON: relation extraction using knowledge graph context in a graph neural network. In: WWW. pp. 1673–1685, 2021.
- [Be13] Berant, Jonathan; Chou, Andrew; Frostig, Roy; Liang, Percy: Semantic parsing on freebase from question-answer pairs. In: EMNLP. pp. 1533–1544, 2013.
- [BH15] Bast, Hannah; Haussmann, Elmar: More accurate question answering on freebase. In: CIKM. pp. 1431–1440, 2015.
- [Bo08] Bollacker, Kurt; Evans, Colin; Paritosh, Praveen; Sturge, Tim; Taylor, Jamie: Freebase: A collaboratively created graph database for structuring human knowledge. In: SIGMOD. pp. 1247–1250, 2008.
- [BOM15] Blanco, Roi; Ottaviano, Giuseppe; Meij, Edgar: Fast and space-efficient entity linking for queries. In: WSDM. pp. 179–188, 2015.
- [CD22] Chatterjee, Shubham; Dietz, Laura: BERT-ER: Query-specific BERT Entity Representations for Entity Ranking. In: SIGIR. pp. 1466–1477, 2022.
- [Ch19] Christmann, Philipp; Saha Roy, Rishiraj; Abujabal, Abdalghani; Singh, Jyotsna; Weikum, Gerhard: Look before you hop: Conversational question answering over knowledge graphs using judicious context expansion. In: CIKM. pp. 729–738, 2019.
- [CSRW22a] Christmann, Philipp; Saha Roy, Rishiraj; Weikum, Gerhard: Beyond NED: Fast and Effective Search Space Reduction for Complex Question Answering over Knowledge Bases. In: WSDM. pp. 172–180, 2022.
- [CSRW22b] Christmann, Philipp; Saha Roy, Rishiraj; Weikum, Gerhard: Conversational Question Answering on Heterogeneous Sources. In: SIGIR. pp. 144–154, 2022.
- [EM10] Erling, Orri; Mikhailov, Ivan: Virtuoso: RDF support in a native RDBMS. In: Semantic Web Information Management. pp. 501–519, 2010.
- [Fe13] Fernández, Javier D; Martínez-Prieto, Miguel A; Gutiérrez, Claudio; Polleres, Axel; Arias, Mario: Binary RDF representation for publication and exchange (HDT). In: Journal of Web Semantics. pp. 22–41, 2013.
- [FS10] Ferragina, Paolo; Scaiella, Ugo: TAGME: On-the-fly annotation of short text fragments (by Wikipedia entities). In: CIKM. pp. 1625–1628, 2010.

- [GSR17] Gupta, Nitish; Singh, Sameer; Roth, Dan: Entity linking via joint encoding of types, descriptions, and context. In: EMNLP. pp. 2681–2690, 2017.
- [HHK15] Hernández, Daniel; Hogan, Aidan; Krötzsch, Markus: Reifying RDF: What Works Well With Wikidata? In: SSWS. p. 32, 2015.
- [Ho11] Hoffart, Johannes; Yosef, Mohamed Amir; Bordino, Ilaria; Fürstenu, Hagen; Pinkal, Manfred; Spaniol, Marc; Taneva, Bilyana; Thater, Stefan; Weikum, Gerhard: Robust Disambiguation of Named Entities in Text. In: EMNLP. pp. 782–792, 2011.
- [JJ19] Joseph, Kevin; Jiang, Hui: Content based news recommendation via shortest entity distance over knowledge graphs. In: WWW. pp. 690–699, 2019.
- [Ko14] Koch, Mitchell; Gilmer, John; Soderland, Stephen; Weld, Daniel S: Type-aware distantly supervised relation extraction with linked arguments. In: EMNLP. pp. 1891–1901, 2014.
- [KSRW21] Kaiser, Magdalena; Saha Roy, Rishiraj; Weikum, Gerhard: Reinforcement Learning from Reformulations in Conversational Question Answering over Knowledge Graphs. In: SIGIR. pp. 459–469, 2021.
- [Li20] Li, Belinda Z.; Min, Sewon; Iyer, Srinivasan; Mehdad, Yashar; Yih, Wen-tau: Efficient One-Pass End-to-End Entity Linking for Questions. In: EMNLP. pp. 6433–6441, 2020.
- [LJ21] Lan, Yunshi; Jiang, Jing: Modeling transitions of focal entities for conversational knowledge base question answering. In: ACL-IJCNLP. pp. 3288–3297, 2021.
- [NW08] Neumann, Thomas; Weikum, Gerhard: RDF-3X: a RISC-style engine for RDF. Proceedings of the VLDB Endowment, pp. 647–659, 2008.
- [Pr21] Pramanik, Soumajit; Alabi, Jesujoba; Saha Roy, Rishiraj; Weikum, Gerhard: UNIQORN: unified question answering over RDF knowledge graphs and natural language text. In: arXiv. 2021.
- [Sa18] Saha, Amrita; Pahuja, Vardaan; Khapra, Mitesh; Sankaranarayanan, Karthik; Chandar, Sarath: Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph. In: AAAI. pp. 705–713, 2018.
- [SKW07] Suchanek, Fabian M; Kasneci, Gjergji; Weikum, Gerhard: YAGO: A core of semantic knowledge. In: WWW. pp. 697–706, 2007.
- [SRA22] Saha Roy, Rishiraj; Anand, Avishek: Question Answering for the Curated Web: Tasks and Methods in QA over Knowledge Bases and Text Collections. Springer, 2022.
- [Su18] Sun, Haitian; Dhingra, Bhuwan; Zaheer, Manzil; Mazaitis, Kathryn; Salakhutdinov, Ruslan; Cohen, William: Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text. In: EMNLP. pp. 4231–4242, 2018.
- [Su20] Sun, Zequn; Wang, Chengming; Hu, Wei; Chen, Muhao; Dai, Jian; Zhang, Wei; Qu, Yuzhong: Knowledge graph alignment network with gated multi-hop neighborhood aggregation. In: AAAI. pp. 222–229, 2020.
- [UJ20] Urbani, Jacopo; Jacobs, Ceriël: Adaptive Low-level Storage of Very Large Knowledge Graphs. In: WWW. pp. 1761–1772, 2020.

- [Ur16] Urbani, Jacopo; Dutta, Sourav; Gurajada, Sairam; Weikum, Gerhard: KOGNAC: Efficient encoding of large knowledge graphs. In: IJCAI. pp. 3896–3902, 2016.
- [Va18] Vashishth, Shikhar; Joshi, Rishabh; Prayaga, Sai Suman; Bhattacharyya, Chiranjib; Talukdar, Partha: Reside: Improving distantly-supervised neural relation extraction using side information. In: EMNLP. pp. 1257–1266, 2018.
- [VK14] Vrandečić, Denny; Krötzsch, Markus: Wikidata: A free collaborative knowledgebase. In: CACM. pp. 78–85, 2014.
- [ZI16] Zhu, Ganggao; Iglesias, Carlos A: Computing semantic similarity of concepts in knowledge graphs. In: IEEE TKDE. pp. 72–85, 2016.
- [Zi17] Ziegler, David; Abujabal, Abdalghani; Roy, Rishiraj Saha; Weikum, Gerhard: Efficiency-aware Answering of Compositional Questions using Answer Type Prediction. In: IJCNLP. pp. 222–227, 2017.



## Session 6



# RAPP: A Responsible Academic Performance Prediction Tool for Decision-Making in Educational Institutes

Manh Khoi Duong<sup>1</sup> Jannik Dunkelau<sup>2</sup> José Andrés Cordova<sup>3</sup> Stefan Conrad<sup>4</sup>



**Abstract:** Due to the increasing importance of educational data mining for the early intervention of at-risk students and the growth of performance data collected in educational institutes, it becomes natural to employ machine learning models to predict student's performances based off prior data. Although machine learning pipelines are often similar, developing one for a specific target prediction of academic success can become a daunting task. In this work, we present a graphical user interface which implements a customizable machine learning pipeline which allows the training and evaluation of machine learning models for different definitions of academic success, e. g., collected credits, average grade, number of passed exams, etc. The evaluation is exported in PDF format after finishing training. As this tool serves as a decision support system for socially responsible AI systems, fairness notions were included in the evaluation to detect potential discrimination in the data and prediction space.

**Keywords:** educational data mining; fairness; decision making; machine learning; academic performance prediction

## 1 Introduction

*Academic performance prediction (APP)* systems can be used to identify *at-risk students* in higher education early on, allowing the university to use resources in a targeted manner to prevent them from achieving poor academic performances. The definition of at-risk students varies as it depends on the context and the purpose of prevention. It can comprise of, e.g., higher chances of dropping out, longer study durations, and worse graduation grades. In this case, the APP system acts as a supporting *artificial intelligence (AI)* system for the university at the institutional level. However, given the impact of such systems onto the student body, social challenges arise. Marcinkowski et al. [Ma20] surveyed the perception of a student body of the use of such AI-based systems and show that APP is viewed as problematic by students as far as their own data and planning are concerned. Furthermore,

<sup>1</sup> Heinrich Heine University, Department of Computer Science, Universitätsstraße 1, 40225 Düsseldorf, Germany  
manh.khoi.duong@hhu.de

<sup>2</sup> Heinrich Heine University, Department of Computer Science, Universitätsstraße 1, 40225 Düsseldorf, Germany  
jannik.dunkelau@hhu.de

<sup>3</sup> Heinrich Heine University, Department of Computer Science, Universitätsstraße 1, 40225 Düsseldorf, Germany  
jose.cordova@hhu.de

<sup>4</sup> Heinrich Heine University, Department of Computer Science, Universitätsstraße 1, 40225 Düsseldorf, Germany  
stefan.conrad@hhu.de

the notion of *fairness-aware machine learning* (FairML) [DL19, Fr19, PS20] becomes an increasingly important topic and also found its way into educational data mining systems [LMZ19, KLM22, HR20, KL20, LQN21, AC19].

Acknowledging these issues, we developed a tool for *responsible academic performance prediction* (RAPP) which tackles two main tasks: it is a tool for (1) academic performance prediction and acts as a (2) decision support system for the social responsibility when employing AI in tertiary education. The first task deals with generating multiple prediction targets and datasets for the prediction of academic performances. The goal of the second task is to find socially acceptable machine learning (ML) models and justify their use from the extensive fairness and interpretability evaluation in the tool. For the full deployment of an AI system to identify at-risk students, ethical aspects and the perception by those affected have to be researched. The fairness and interpretability evaluation plays a supportive role to disregard or regard certain ML models by, e. g., checking whether they comply with student's perception of discrimination or do not discriminate through socio-demographic features.

The source code of the RAPP tool is published under the MIT License and available online at <https://github.com/hhu-rapp/rapp-tool>.

## 2 Related Systems

Our proposed tool combines functionalities from two different research communities: (educational) data mining and fairness assessment. In this section, we will briefly present selected tools already available from either community.

RapidMiner [HK16], Orange [De13], and WEKA [Ha09]—to name a few—are data mining tools with a graphical user interface (GUI) just as the proposed tool in this paper. The aforementioned tools mostly include data visualization, pre-processing, feature selection, clustering, classification, regression, and evaluation metrics. The tools are modular, meaning the pipeline and its specific configurations are highly modifiable. Their aim is to enable data mining practitioners the comparison of machine learning models on custom datasets without having to write code themselves.

Although not as comprehensive and powerful, tools that were explicitly developed for educational data exist as well. They predominantly focus on a specific dataset that was provided by a particular educational institute. Especially, they analyze and predict several students' data such as programming grades [Ba16], examinations of the final school year [LMP16], students' contributions in group programming [SA20], or students' written feedback [Gr20].

Fairness and transparency in machine learning have become more important in recent years due to the awareness of potential mistreatment of AI over different demographic groups [DL19, Fr19, PS20]. As a response, authors began developing tools to audit the

fairness of an ML system and to produce bias reports, to guide the selection process of a fitting fairness metric, or to apply intervening methods to reduce exhibited bias. Examples for such tools are Aequitas [Sa18], FairSight [AL19], Fairlearn [Bi20], or Fairness Compass and Fairness Library [RD22]. These topics have also been recognized by the educational data mining (EDM) community lately. To name some, Hu and Rangwala [HR20] and Kizilcec and Lee [KL20] consider prejudice and unfairness where Le Quy and Ntoutsis [LQN21] and Alonso and Casalino [AC19] acknowledge the explainability of the used models in EDM.

For the proposal, the RAPP tool aims to take on the preliminary works and combine functionalities from both communities: It is a data mining tool for educational data that includes fairness examinations and interventions to address responsibility when employing AI in educational institutes.

### 3 RAPP Tool

Making it possible to easily create various datasets from a single database with desired features and labels to train, save, and evaluate machine learning algorithms is the aim of the developed tool. For this, the GUI provides an intuitive way to load a particular SQLite database or a CSV file<sup>5</sup> and specify the initial settings for the machine learning pipeline. The demanded features and target labels can be derived by querying the database. Several settings are detected automatically such as the prediction type (classification, regression), the target variable (last column by default), and categorical features. The supported estimators for classification are *decision trees*, *random forest*, *support vector machine*, *naive bayes*, and *logistic regression* and for regression *linear regression*, *elastic net*, *bayesian ridge*, *decision tree regressor*, and *kernel ridge*. An *artificial neural network* with two hidden layers is also available for both of these task types. Experienced users can modify the configuration for their needs. Fig. 1 displays the user interface for the settings.

In the following, we will outline the two main uses and functionalities of the RAPP tool: APP and supporting the decision-making process whilst designing a responsible APP.

#### 3.1 Academic Performance Prediction

##### 3.1.1 Pipeline

At the front of the RAPP tool lies the ability to setup and train APP models over the implemented ML pipeline. The pipeline is outlined in Fig. 2. First, the pipeline's settings have to be specified. This includes the selection of a dataset to use for training as well as picking the ML algorithms to train.

---

<sup>5</sup> The CSV file is treated as a database.

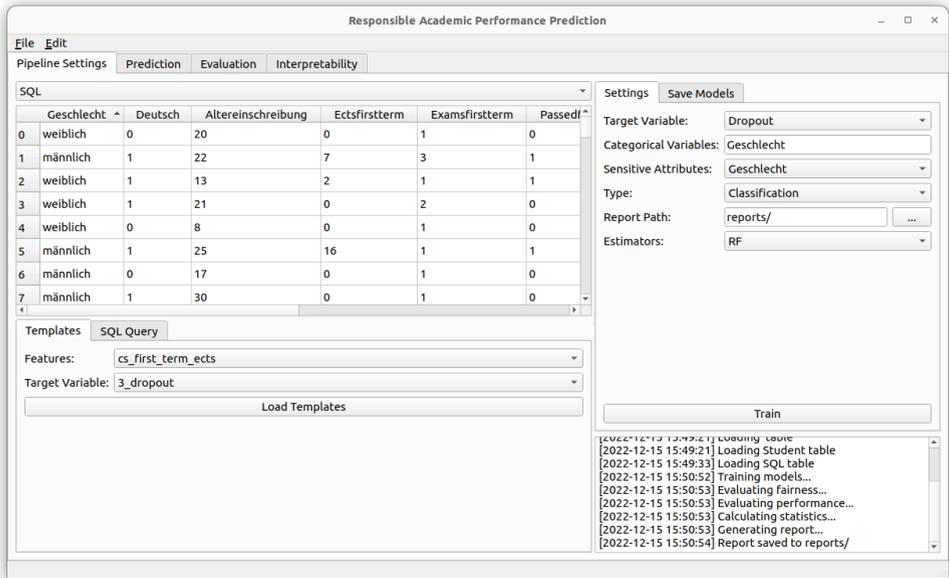


Fig. 1: RAPP's Pipeline Settings Interface, 2022.

The data are queried over an SQLite database. While advanced users can enter custom queries on the database for feature engineering and feature selection, predefined feature and label sets were added for the given academic database to comfortably reuse and combine them in any desired pairing. The user can select, for instance, features such as credit points, grades, or number of passed exams, and target labels such as final GPA, achieved credits until semester  $x$ , or study duration. To ease working with different sets of features and labels we implemented an SQL templating engine which produces the final query based on the user's selections for a feature and a label set. This avoids combinatoric explosion which would arise if each feature-label pair's SQL query had to be implemented manually. The queried database then acts as a dataset for the machine learning pipeline.

Once the dataset is obtained, the features go through the pre-processing step of one-hot encoding any categorical features. After this, the data is split into training (80 %) and test (20 %) data.

Each of the user's selected models are trained on the training data. We also evaluate the performance over the training data via 5-fold cross-validation to capture how robust the models behave during training. The training concludes in an evaluation over various performance metrics as well as fairness metrics. Fairness is also audited directly over the dataset as well. The evaluation results are saved into a detailed PDF report file containing information over the demographics of the dataset as well as the performance and fairness results of each trained estimator.

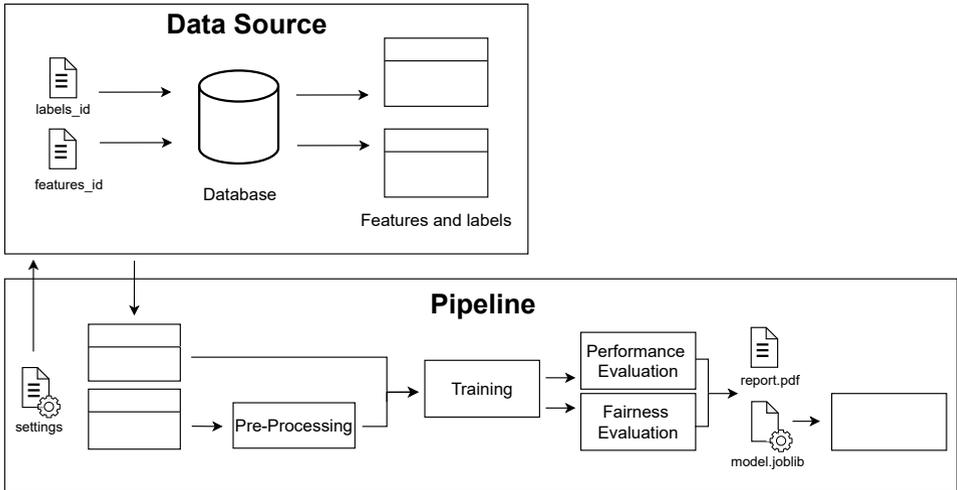


Fig. 2: RAPP’s Machine Learning Pipeline, 2022.

After the trained models are evaluated, the users can decide which models they want to save in order to use them later to predict on new data.

### 3.1.2 Prediction

To tackle the task of identifying at-risk students early, this tool includes a prediction interface as shown in Fig. 3. This interface enables the user to make predictions based on individual student’s academic data. The user can then identify students who are more likely to benefit from the institution’s support programs.

In order to predict the students’ performances, new data from students as well as compatible models, i. e., models that have been trained with the same features, are required in the prediction interface. It is possible for the user to load various models trained for different target variables to predict several targets from the same features simultaneously. Once new data and selected models are loaded into the GUI, the features go through a pre-processing step and are then fed into the loaded models for the prediction. Fig. 3 shows an example of multiple targets being predicted with the data of a single student.

After the prediction has been run, the interface updates and displays the predictions of the models for each of the selected targets. It is also possible to load multiple models for one specific target to employ ensemble learning. In case of classification, we apply majority voting whereas in regression tasks the mean of the predicted values is used.

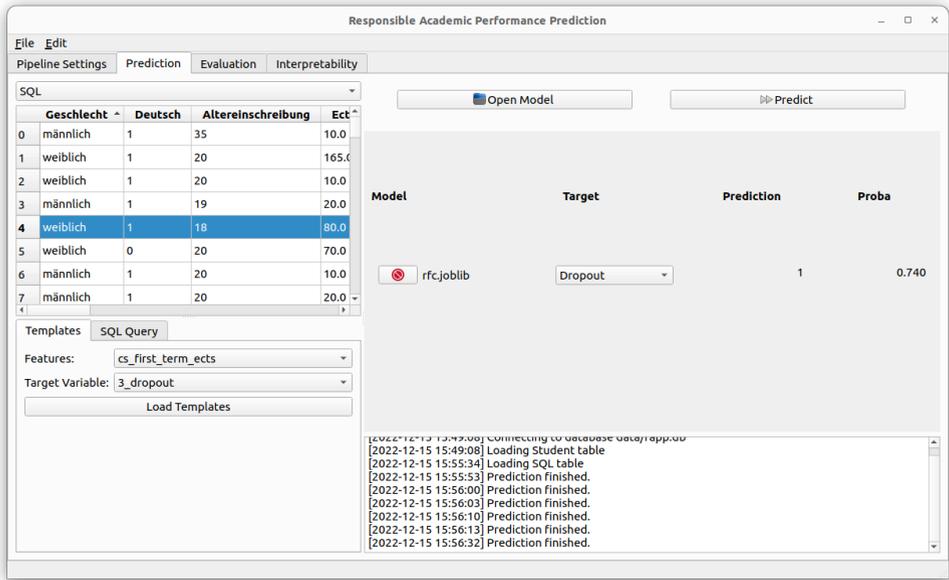


Fig. 3: RAPP's Prediction Interface, 2022.

### 3.2 Decision Support System

The tool acts as a decision support system by providing the user statistical insights of the dataset as well as an extensive evaluation of the models' performance and fairness. The models are automatically evaluated on the training and test data as they progress through the pipeline. The evaluation is displayed in the GUI, part of it is shown in Fig. 4, and is also generated as a  $\text{\LaTeX}$  report, that is automatically compiled as a PDF file.

**Dataset.** The dataset tab contains a contingency table that displays the label  $y \in \{0, 1\}$  and the sensitive attribute. This allows the user to comprehend the relationship between the sensitive attributes and the students' performances.

**Performance Metrics.** As for stability reasons, the evaluation for the training data is always done with 5-fold cross-validation. The type of task that was selected beforehand determines the suitable metrics. Classification metrics included in the tool are *accuracy*, *balanced accuracy*,  $F_1$ , *recall*, *precision*, and *area under ROC*. As for regression metrics, the tool implements *mean absolute error*, *mean squared error*, *max error*, and  $R^2$ .

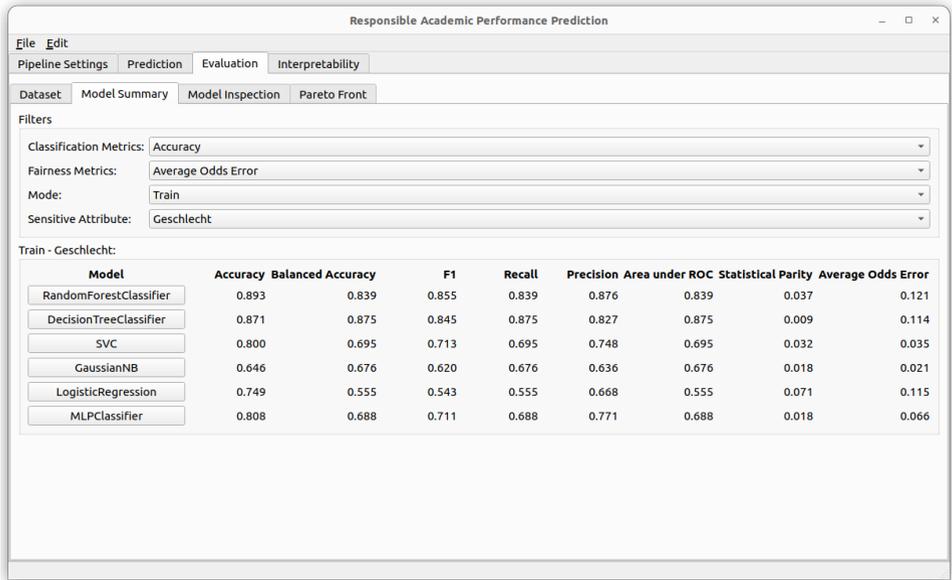


Fig. 4: RAPP’s Decision Support System Interface, 2022.

**Fairness Notions.** The fairness of the models’ predictions is assessed with regard to the sensitive attributes in order to detect potential discrimination. Similarly to the performance metrics, the notions are determined by the task type. Classification tasks implement *statistical parity*, *predictive equality*, and *equality of opportunity* [DL19, BHN19]. While statistical parity is one of the most commonly used fairness notions, recent work suggests a focus on *equalized odds* (requires predictive equality as well as equality of opportunity) as the go-to notion for APP systems [DD22]. Accordingly, the tool integrates *average odds error* [Be18] which quantifies equalized odds. For regression tasks we use the *individual fairness* and *group fairness* notion as introduced by Berk et al. [Be17].

To measure fairness criteria in classification, we use the absolute difference of the outcomes between two groups. Generally, a lower value describes less discrimination. Because group sizes greater than two (non-binary genders, multiple nationalities) might occur in the dataset, we use the maximum value of the absolute differences between all group pairs [Ž17]. This measures the maximal discrimination a classifier has achieved between two groups.

**Pareto Front: Performance and Fairness Trade-Off.** Due to the existence of a performance and fairness trade-off [BFT12], the trade-off can be visually examined in order to select the best trained models to use for predictions. The Pareto-efficient models, i. e., models that optimize both a particular performance metric and fairness measure, can then be identified. The fairness tab includes scatter points of the selected models in a

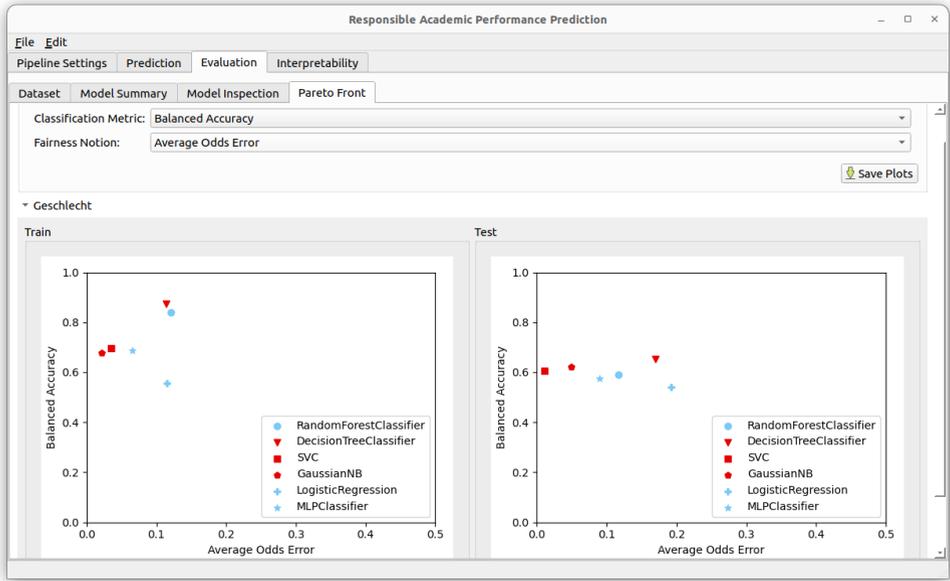


Fig. 5: RAPP's Pareto Front Evaluation, 2022.

performance-fairness plot (see Fig. 5). The Pareto front, i. e., the set of all Pareto-efficient models [JS08], is shown in a different color to differentiate them from Pareto-dominated points. Pareto-efficient models are displayed in red whereas Pareto-dominated models are displayed in lightblue. This visualization limits the decision-making space for the user as only Pareto-efficient models are of interest. Because Pareto optimal solutions are first shown and the decision-maker selects her/his preferred model afterwards, this is a posteriori method in decision-making.

In classification we aim to maximize the performance metric whereas a maximization of the performance in regression corresponds to minimizing the error. For contextual conveniences, we maximize the negative error in regression to yield for the same plot.

## 4 Case Study

The RAPP tool is developed as part of a research project concerning itself with designing a socially responsible framework on how to approach APP in higher education. For this, we conducted a case study over data given to us by the Heinrich Heine University Düsseldorf. The case study was concerned mostly with probing of which prediction tasks show most-promising performances and to estimate possible algorithmic fairness problems. Hereby, the prediction tasks differed in their combination of input features as well as at-risk definition for prediction.

| Input               | Prediction |         |      |
|---------------------|------------|---------|------|
|                     | Dropout    | MA Adm. | SDS  |
| ECTP + Exam stats   | 0.65       | 0.63    | 0.67 |
| Grades + Exam stats | 0.68       | 0.67    | 0.61 |
| Specific modules    | 0.74       | 0.62    | 0.63 |

Tab. 1: Overview of exemplary training results over CS students in their first semester. Displaying the best performing balanced accuracy achieved by any trained model over combinations of selected feature sets and the prediction of student dropouts, master program admission (MA Adm.), and finishing in standard duration of study (SDS).

As we were interested in any combination of these predefined features and prediction goals, the RAPP tool was a great help in leveraging the combinatorial explosion problem into a manageable set of selectable templates, allowing us to quickly train and store models for each combination. Fig. 4 displays one such training result as reported within the tool, allowing comparison of the trained models over various performance and fairness measures. Tab. 1 shows exemplary results conducted with the RAPP tool over computer science (CS) students after their first semester.

## 5 Limitations and Future Work

Since the tool is still in development, new opportunities for future improvements present themselves constantly. These enhancements include changes to the tool’s architecture, as well as making the prediction process more transparent to the user.

The tool as it currently is comes with SQL templates designed for our database in use. However, in order to allow other educational institutions to target at-risk students, the tool allows to write a different set of SQL templates and to load any SQLite database, making the tool essentially database agnostic. Still, this requires proficiency with writing SQL queries and modulating them into the templating engine, a skill that end users might not have. Here, the ease of use could be enhanced.

While allowing to inspect potentially exhibited discrimination by the trained models, it is not yet possible to train models with fairness-interventions in mind. In the future we would like to incorporate ways to train models with fairness-accounting measures such as pre-, in-, and postprocessing [DL19, Fr19, PS20]

## 6 Discussion

The tool helps in investigating which fairness constraints are met by any trained model and thus guides the user in their decision making of which model to employ, but by no means does the tool alone help achieving the overall goal.

Approaching RAPP includes to find a suitable definition for algorithmic fairness by involving both, the institute's stakeholders as well as the affected student body [KLM22]. While the notion of equalized odds seems to be a desirable fairness constraint [DD22], the student body appears to favor demographic parity [Ma20]. Further, the potential damage caused by misclassifications needs to be carefully considered. All these above points are not meant to be resolved by the RAPP tool but rather need to be part of the conceptualization when planning to employ such a system *before* actual employment of the system takes place. However, the RAPP tool helps to investigate whether potential concerns are dealt with appropriately by the trained models or not.

## 7 Conclusion

In this paper, we presented the RAPP tool for developing responsible academic performance prediction systems. The tool tackles two main tasks: designing, training, and analyzing different APP tasks, and acting as a decision support system for selecting the best suited models in a fairness-sensitive and socially responsible context.

For the setup of APP tasks, the concurrent design and direct comparison of different tasks, i. e., different input features and target labels, was a main objective as the definition of academic performances differ depending on the viewpoints of users, the student body, or the application context. In order to assist the user which model or models are socially responsible when being employed to target interventions at at-risk students, extensive performance and fairness metrics are included. The metrics are viewable in the GUI itself but are also automatically exported to a PDF file. Assessing fairness metrics and highlighting the Pareto front of classical performance metrics and achieved fairness parities guides the user in the decision-making process of finding the most suitable model for their desired task.

Overall, the tool provides an interface to non-machine learning engineers to train, evaluate, and employ models in the APP domain by providing a simplified ML pipeline configuration and highlighting crucial trade-offs of the model accuracy vs. fairness, rendering responsible APP systems a step more accessible and approachable to everyone.

## Acknowledgments

This work was supported by the Federal Ministry of Education and Research (BMBF) under Grand No. 16DHB4020.

## Bibliography

- [AC19] Alonso, José M; Casalino, Gabriella: Explainable artificial intelligence for human-centric data analysis in virtual learning environments. In: International workshop on higher education learning methodologies and technologies online. Springer, pp. 125–138, 2019.
- [AL19] Ahn, Yongsu; Lin, Yu-Ru: FairSight: Visual analytics for fairness in decision making. *IEEE transactions on visualization and computer graphics*, 26(1):1086–1095, 2019.
- [Ba16] Badr, Ghada; Algobail, Afnan; Almutairi, Hanadi; Almutery, Manal: Predicting students' performance in university courses: a case study and tool in KSU mathematics department. *Procedia Computer Science*, 82:80–89, 2016.
- [Be17] Berk, Richard; Heidari, Hoda; Jabbari, Shahin; Joseph, Matthew; Kearns, Michael; Morgenstern, Jamie; Neel, Seth; Roth, Aaron: A convex framework for fair regression. arXiv preprint arXiv:1706.02409, 2017.
- [Be18] Bellamy, Rachel K. E.; Dey, Kuntal; Hind, Michael; Hoffman, Samuel C.; Houde, Stephanie; Kannan, Kalapriya; Lohia, Pranay; Martino, Jacquelyn; Mehta, Sameep; Mojsilovic, Aleksandra; Nagar, Seema; Ramamurthy, Karthikeyan Natesan; Richards, John T.; Saha, Diptikalyan; Sattigeri, Prasanna; Singh, Moninder; Varshney, Kush R.; Zhang, Yunfeng: AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. CoRR, abs/1810.01943, 2018.
- [BFT12] Bertsimas, Dimitris; Farias, Vivek F; Trichakis, Nikolaos: On the efficiency-fairness trade-off. *Management Science*, 58(12):2234–2250, 2012.
- [BHN19] Barocas, Solon; Hardt, Moritz; Narayanan, Arvind: Fairness and Machine Learning. fairmlbook.org, 2019. <http://www.fairmlbook.org>.
- [Bi20] Bird, Sarah; Dudík, Miro; Edgar, Richard; Horn, Brandon; Lutz, Roman; Milan, Vanessa; Sameki, Mehrnoosh; Wallach, Hanna; Walker, Kathleen: Fairlearn: A toolkit for assessing and improving fairness in AI. Microsoft, Tech. Rep. MSR-TR-2020-32, 2020.
- [DD22] Dunkelau, Jannik; Duong, Manh Khoi: Towards Equalised Odds as Fairness Metric in Academic Performance Prediction. In: 2nd Workshop on Fairness, Accountability, and Transparency in Educational Data. July 2022.
- [De13] Demšar, Janez; Curk, Tomaž; Erjavec, Aleš; Črt Gorup; Hočevar, Tomaž; Milutinovič, Mitar; Možina, Martin; Polajnar, Matija; Toplak, Marko; Starič, Anže; Štajdohar, Miha; Umek, Lan; Žagar, Lan; Žbontar, Jure; Žitnik, Marinka; Zupan, Blaž: Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*, 14:2349–2353, 2013.
- [DL19] Dunkelau, Jannik; Leuschel, Michael: Fairness-Aware Machine Learning: An Extensive Overview. Working paper, available at <https://www3.hhu.de/stups/downloads/pdf/fairness-survey.pdf>, October 2019.
- [Fr19] Friedler, Sorelle A.; Scheidegger, Carlos; Venkatasubramanian, Suresh; Choudhary, Sonam; Hamilton, Evan P.; Roth, Derek: A comparative study of fairness-enhancing interventions in machine learning. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. ACM, jan 2019.
- [Gr20] Grönberg, Niku; Knutas, Antti; Hynninen, Timo; Hujala, Maija: An online tool for analyzing written student feedback. In: Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research. pp. 1–2, 2020.

- [Ha09] Hall, Mark; Frank, Eibe; Holmes, Geoffrey; Pfahringer, Bernhard; Reutemann, Peter; Witten, Ian H: The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
- [HK16] Hofmann, Markus; Klinkenberg, Ralf: *RapidMiner: Data mining use cases and business analytics applications*. CRC Press, 2016.
- [HR20] Hu, Qian; Rangwala, Huzefa: *Towards Fair Educational Data Mining: A Case Study on Detecting At-Risk Students*. International Educational Data Mining Society, 2020.
- [JS08] Jin, Yaochu; Sendhoff, Bernhard: Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415, 2008.
- [KL20] Kizilcec, René F.; Lee, Hansol: *Algorithmic Fairness in Education*. arXiv, 2020.
- [KLM22] Keller, Birte; Lünich, Marco; Marcinkowski, Frank: How Is Socially Responsible Academic Performance Prediction Possible? In: *Strategy, Policy, Practice, and Governance for AI in Higher Education Institutions*, pp. 126–155. IGI Global, may 2022.
- [LMP16] Livieris, Ioannis; Mikropoulos, Tassos; Pintelas, Panagiotis: A decision support system for predicting students' performance. *Themes in Science and Technology Education*, 9(1):43–57, 2016.
- [LMZ19] Loukina, Anastassia; Madnani, Nitin; Zechner, Klaus: The many dimensions of algorithmic fairness in educational applications. In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, pp. 1–10, August 2019.
- [LQN21] Le Quy, Tai; Ntoutsis, Eirini: Towards fair, explainable and actionable clustering for learning analytics. In: *EDM*. 2021.
- [Ma20] Marcinkowski, Frank; Kieslich, Kimon; Starke, Christopher; Lünich, Marco: Implications of AI (un-)fairness in higher education admissions. In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. ACM, jan 2020.
- [PS20] Pessach, Dana; Shmueli, Erez: *Algorithmic Fairness*. volume abs/2001.09784, 2020.
- [RD22] Ruf, Boris; Detyniecki, Marcin: A Tool Bundle for AI Fairness in Practice. In: *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. pp. 1–3, 2022.
- [Sa18] Saleiro, Pedro; Kuester, Benedict; Stevens, Abby; Anisfeld, Ari; Hinkson, Loren; London, Jesse; Ghani, Rayid: *Aequitas: A Bias and Fairness Audit Toolkit*. arXiv preprint arXiv:1811.05577, 2018.
- [SA20] Sandee, Jan Jaap; Aivaloglou, Efthimia: Gitcanary: A tool for analyzing student contributions in group programming assignments. In: *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. pp. 1–2, 2020.
- [Ž17] Žliobaitė, Indrė: Measuring discrimination in algorithmic decision making. *Data Mining and Knowledge Discovery*, 31:1060–1089, 2017.

# Semantic Watermarks for Detecting Cheating in Online Database Exams

Stefan Brass,<sup>1</sup> Alexander Hinneburg<sup>2</sup>

## Abstract:

Due to the COVID-19 pandemic, we were forced to conduct two exams for a database course as online exams. An essential part of the exams was to write non-trivial SQL queries for given tasks. In order to ensure that cheating has a certain risk, we used several techniques to detect cases of plagiarism. One technique was to use a kind of “watermarks” in variants of the exercises that are randomly assigned to the students. Each variant is marked by small discrimination points that need to be included in submitted solutions. Those markers might go through undetected when a student decides to copy a solution from someone else. In this case, the student would reveal to know a “secret” that he cannot know without the forbidden communication with another student. This can be used as a proof for plagiarism instead of just a subjective feeling about the likelihood of similar solutions without communication. We also used a log of SQL queries that were tried during the exam.

**Keywords:** SQL; Plagiarism; Online Exams; Cheating; Academic Integrity

## 1 Introduction

The COVID-19 pandemic made classical, proctored exams for large classes impossible. Therefore online exams suddenly became interesting in spite of doubts that students might cheat. We limit our discussion to the case when only the communication between students during the exam must be prevented. Otherwise the exam is “open book”.

It is often argued that the grade distribution of the online exam results is not very different from normal exams. Therefore, the percentage of good marks achieved by cheating cannot be very large. However, this argument disregards the possibility that students participate in the exams who are not well prepared. Maybe, they would have learnt more, or taken a later exam, if they had not relied on the possibility of cheating. Conversely, the first online exams were prepared without much experience, and the exam setting was also new for the students. It cannot be excluded that some good students got not so good grades. Whoever is responsible for an exam has the obligation to create a setting in which the honest students are not the fools.

The paper [Ja21] reports the results of an online questionnaire answered by 1608 German students. 45.9% of the participants admitted „Exchanging ideas with others about possible

---

<sup>1</sup> Martin-Luther-University, Computer Science, Germany brass@informatik.uni-halle.de

<sup>2</sup> Martin-Luther-University, Computer Science, Germany hinneburg@informatik.uni-halle.de

answers during an examination“ at least once in an online exam. The studies cited in [Di03] are in the same range or higher (but may include homework assignments). The paper [DS17] proposes statistical tests to check whether cheating occurred in an exam based on the grade distribution, and also using the normal dependency of the grade on attendance of classes and the number of completed homework assignments. The results showed that cheating did occur in most of the investigated online exams. In [Eh21], conversely bad exam results of some students with good performance in the developed RA/SQL tutoring system were noticed, leading to the question of plagiarism in the homeworks. [CI20] mentions exam questions appearing on commercial websites and being rapidly answered by tutors of these services (still during the exam). See also [LC15]. In our online exam in February 2022, we detected 14 students who participated in the exchange of solutions (out of 114, i.e. 12%).

Approaches that have been used to prevent cheating in online exams include software solutions like exam browsers and video supervision [FMG19, Ba17]. Since the student is at home and can prepare the setup, software solutions and video supervision do not really prevent cheating [Bi15]. Other counter measures against cheating include the preparation of large question pools from which questions are randomly assigned to individual exams. This prevents cheating between particular pairs of students who agreed to close collaboration during online exams. However, question pools have limited effects in larger collaborative groups of students. Furthermore, building such a question pool is a lot of work. Since questions of an online exam can easily be copied by the students, they also cannot be reused in the near future.

It is important that there is at least a certain risk involved in cheating to counter the negative consequences for academic integrity. Discussions among students in anonymous online forums show that most students thought that the risk is near zero and it would be stupid not to use the chances offered by online exams [An22]. A good method against cheating in online exams should fulfill the following criteria: (a) the method should increase the risk of being caught for the involved students, (b) it should increase the time needed for the students to cheat in an undetected way, (c) it should be easily applicable and the lecturer, who designs the exams, should not need to spend much additional time for creating and checking the exercises.

We propose a method called semantic watermarks that fulfills the above mentioned criteria. The method works for all types of tasks, where a written answer is required that must include specific semantic pieces from the task description, e.g. programming and design tasks.

The remainder of the paper is structured as follows: In Section 2, we discuss related work about cheating in online exams. Section 3 introduces the semantic watermark method in a formal way, discusses practical issues and ways to strengthen semantic watermarks. Section 4 describes applications of the method in real exams. The results on SQL queries are compared with those from a log-file analysis in Section 5. Section 6 gives some additional practical advice on what to do before and after the exam. Section 7 concludes the paper.

## 2 Related Work

So far, there has not been a lot of research specifically on SQL plagiarism detection (e.g., the literature overview [NJK19] contains no entry for SQL). The paper [RC05] suggests some methods for detecting “similar” queries, like comparing the queries with ignoring case and non-essential white space, and a “histogram” method that looks at specific seldom characteristics of the query, such as trailing (invisible) whitespace in the lines, or inconsistent case in keywords. The paper contains important ideas, but unfortunately, as also [KS19] notices, not enough details are given so that one can exactly reproduce the method. [KS19] have implemented the first two methods of [RC05], and tested them for a coursework dataset (with lots of plagiarism) and an exam dataset (with basically no plagiarism). It turned out that even in the exam dataset, many queries were classified as being plagiarized (and are therefore “false positives”). As [KS19] noted, this might be due to the relative short length of queries in both datasets. They suggest that for queries of over 200 characters, the algorithms might work, but they have not enough data to reach a definite conclusion. The queries in our exam have an average length of 250 characters, and 62% of the queries are at least 200 characters. Nevertheless, in our tests with such similarity-based methods, either only very few of the actual cheaters are found, or so many queries are detected as similar that the actual cheaters are like a needle in a haystack (assuming that the methods presented in this paper give a good approximation of the “actual cheaters”).

The teaching situation in [SSS18] is special, because the students have to develop also a database schema, which gives a much larger space of possible solutions. The methods of [MJ08, Si13] are applicable only for Microsoft Access (at least, students need to submit binary database files). The method of [DH05] also uses a “watermark” for detecting plagiarized Java programs, but needs to be able to change files on the computer of the student. A well-known plagiarism checker for Java programs is JPlag [PMP02]. A general literature overview on cheating in online exams is [NMA22]. Watermarks for relational data were studied, e.g., in [AHK03, La08, GA11]. During manual grading, copied solutions are often detected because of the same strange mistakes. For classifications of SQL mistakes and style/performance problems, we refer to [BG06, TSV18, MAF21].

Different versions of exercises have been used for a long time, and there are also techniques where randomly chosen numbers are inserted into the task description of an exercise. Our “watermark method” works with parameterized task descriptions, too. However, we use this for detecting cheating attempts. Before, the main goal was that students get no points for an exercise if they copy a solution, because ideally, every student has a different exercise. One often cannot distinguish whether the student copied a solution or whether he simply made a mistake. In our case, the idea is that the watermarks appear in the submitted solution and can be used as a proof for cheating. Of course, if it was visible that a submitted solution was for a different exercise variant, a cheating attempt was detected even without having a fancy name for this method. However, our contribution still is to systematically define this method, give suggestions for not eye-catching differences, test the method in several exams, and compare it with other methods for detecting plagiarized SQL-queries.

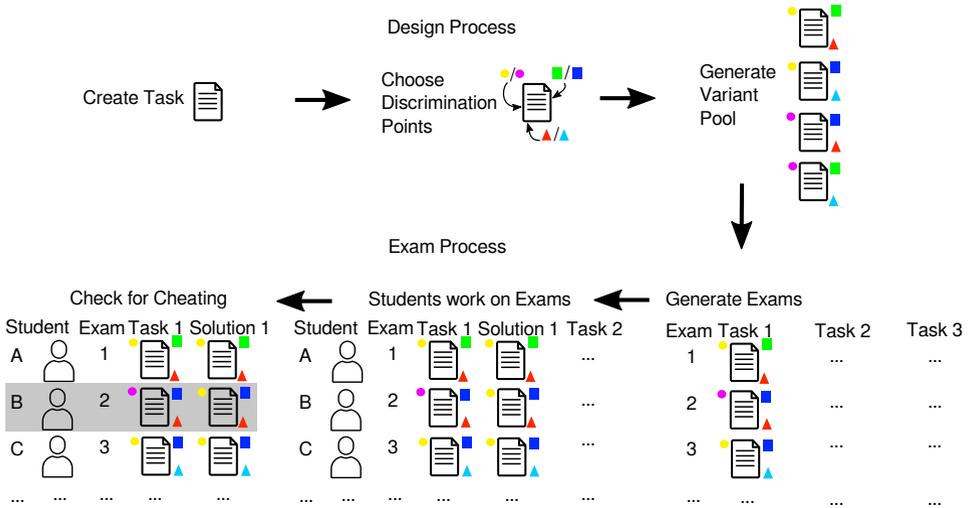


Fig. 1: Overview of the semantic watermark method

### 3 Semantic Watermarks

The method of semantic watermarks is sketched in Figure 1. At the beginning, the lecturer designs a specific task, e.g. the creation of an SQL query to retrieve specific data that is described in the task’s text. In a next step, one or more discrimination points are chosen, which are short pieces in the task text that can be replaced by different values to generate other but semantically similar variants of the task. In Figure 1, the discrimination points are represented by different icon shapes and the different values for the discrimination points are represented by different colors. A discrimination point could be a constant that needs to be used in the WHERE clause. We discuss several criteria how to choose good discrimination points and the corresponding values. The discrimination points allow the generation of a pool of variants of the task that differ only at those points. Most online exam software packages like ILIAS or MOODLE allow to use such questions pools to randomly generate exams. A particular student sees only one of the variants of the task in the exam and submits a solution based on that particular textual task description. The discrimination points are designed in such a way that a complete solution needs to include values for all discrimination points mentioned in the task. Therefore, a student could be identified to cheat, when the solution does not match in the values of all discrimination points with the corresponding task description. In this case, the student would reveal to know information that he or she cannot know without the forbidden communication with another student (because the value was not contained in his/her exam version). In Figure 1, student B cheated in the submitted solution, because the yellow dot did not appear in corresponding task description.

### 3.1 Definition and Preconditions

The formal definition of the semantic watermarks method starts with a given task  $t$  that is described by a text or a picture. Given a task  $t$ , a number of  $n$  discrimination points needs to be chosen.

Each discrimination point is associated with a set of several values that are called the domain  $D_i$  for the respective point,  $1 \leq i \leq n$ . Each domain needs to have at least two different values:  $|D_i| \geq 2$  for  $1 \leq i \leq n$ . The values in a particular domain should identify the corresponding discrimination point. However, NULL values are not allowed in those domains. The choice of discrimination points for the task  $t$  defines a mapping  $\tau(d_1, \dots, d_n)$  that is called the parameterized task. A parameterized task is a mapping that takes a vector  $\vec{d} = (d_1, \dots, d_n)$  from the cross product of the respective domains  $D_1 \times \dots \times D_n$  of the discrimination points. Such a vector  $\vec{d}$  is called a discrimination vector. A parameterized task outputs a task description, which can be a text or a picture. Usually, the original task  $t$  equals the output of  $\tau$  for some discrimination vector  $\vec{d}$ . The pool of variants of the parameterized task  $\tau$  is generated by choosing a set of discrimination vectors and mapping them to texts or images via  $\tau$ .

Each student sees the output  $t'$  of the parameterized task  $\tau$  based on a particular discrimination vector. Based on this description she or he develops a solution  $s$  for  $t'$ . The solution  $s$  needs to include values for the discrimination points that forms another discrimination vector that is from the cross product of extended domains  $\hat{D}_1 \times \dots \times \hat{D}_n$  with  $\hat{D}_i = D_i \cup \{\text{NULL}\}$  for  $1 \leq i \leq n$ . The original domains are extended by NULL values because a student may not submit a complete solution. Thus, certain aspects including some values for discrimination points might be missing. Given the discrimination vector  $\vec{v}^t$  responsible for the task description shown to the student and the discrimination vector  $\vec{v}^s$  included in his/her solution, a cheating attempt is detected if and only if there is a discrimination point  $i$ ,  $1 \leq i \leq n$ , such that  $v_i^t \neq v_i^s$  and  $v_i^s \neq \text{NULL}$  (i.e. the values are both defined and different).

The definition of semantic watermarks requires at least one discrimination point. However, semantic watermarks with only a single discrimination point are not very useful, even when the number of values for this point is large. It is likely that students, who want to cheat, would spot this single point by chance and use the correct value for their variant of the task. Therefore, multiple discrimination points are preferable and would lower this chance. The number of values per discrimination point does not need to be large. In fact two different values would suffice, which would create binary discrimination points.

When a pool of variants of a task is generated, the discrimination vectors of the semantic watermark should be chosen from the cross product of the domains  $D_1 \times \dots \times D_n$  in such a way that the discrimination vectors have pairwise Hamming distances equal or larger than two. If the value for more than one discrimination point is wrong, the evidence is very strong. Furthermore, the larger the minimal pairwise Hamming distance, the lower the risk that cheating students find all discrimination points.

### 3.2 Practical Issues

The design of effective discrimination points requires to balance two conflicting goals: (i) the discrimination points need to have distinct and clearly different values that could not be mixed by chance and (ii) the discrimination points should not be striking and eye-catching to a casual observer. Furthermore, since students were shown the expected result of the query with respect to a given database state and could try every query, we choose the discrimination point values in such a way that they do not influence the query result for this particular state (at least not in an eye-catching way).

A simple case of a discrimination point is a condition of the form  $A > c$ . Then we could choose several constants  $c_1 < c_2 < \dots < c_m$  as differentiating values, such that the example table contains no data values between  $c_1$  and  $c_m$ . The constant  $c_i$  of a different exercise variant could be a good indicator of cheating if: (1) The constants are sufficiently different, i.e. differencing characters should not be adjacent on the keyboard, and ideally the constants should differ in more than one character. (2) They should also not appear in the given database state, so that the student cannot know them. The differences are not so easy to spot if first and last character are the same. Also many-to-many relationships offer possibilities for differences in the query without differences in the result. E.g., if the query asks for the courses taken by a particular student, it might be that two students took the same courses.

“Redundant conditions” are another possibility to design discrimination points. Suppose the main query condition is  $C$ . Now variant  $j$  of the exercise ( $1 \leq j \leq m$ ) could require in addition that the data satisfies  $C'_j$ . Therefore, the correct query condition is  $C \wedge C'_j$ . But if the given database state does not contain data satisfying  $C \wedge \neg C'_j$ , this condition does not change the answer in the test state. However, there is a tradeoff: This also means that the given example state does not test for the missing condition  $C'_j$ .

Another option to construct discrimination points are “ORDER BY” specifications, where two different result columns happen to give the same sequence of table rows. For instance, in one variant we could ask the students to order employees by HIREDATE and in another variant by EMPNO, when both cases lead to the same order of employees.

Result column names could be used as discrimination points (we used this a lot). In a similar way, the task might ask for string constants as part of the SELECT-clause that can be used as discrimination points. For instance, the students might have to construct a string with some fixed part and a data value with the string concatenation operator `||`. Or the query might require the UNION operator, and a discrimination point uses a fixed value for a result column (e.g., some kind of summary row). However, the differences between the different values of such a discrimination point should not be too small.

We checked for cheating attempts first with `grep`, then with SQL scripts using `LIKE` and `SIMILAR TO`. In both cases, the values for the discrimination points must be quite specific. Of course, with a full SQL parser (or manual checking), the range of possible values increases.

### 3.3 Risks, Costs, and Countermeasures

Even when a student is able to spot all discrimination points, this operation consumes some amount of time. Thus, cheating still comes at a certain cost. This cost might be larger than expected because students, who did not explicitly prepare for this method, will probably exchange only solutions to the exercises. Checking the solution of another person for small differences to the assigned task is much harder than comparing the task descriptions.

If the students know in advance that this method will be applied, they will also exchange the task descriptions and use programs like `diff` to spot the discrimination points. An effective counter measure for this behavior would be to render the task descriptions as non-textual images that could not be automatically compared by `diff` programs. Alternatively, one could add many unessential differences, such as doubling some spaces between words.

## 4 Applications in Online-Exams

We applied semantic watermarks in three exams for an introductory database course — two online exams with over a hundred participants each (in March 2021 and February 2022) and one smaller proctored exam (repetition exam in July 2022).

Besides the watermarks, there were two obviously different versions of each SQL exercise in both online exams. Probably, students expected different exercise versions, and maybe, after having detected the obvious differences, some did not search for additional, much smaller differences.

For space reasons, we show only the results of the exam in February 2022 (114 participants). We detected 7 cheating attempts of 6 students using the watermarks method:

| Student | Task             | DPs | Hamming Distance | DPs wrong | DPs adapted | DPs omitted | Group Size |
|---------|------------------|-----|------------------|-----------|-------------|-------------|------------|
| A       | SQL (NOT EXISTS) | 3   | 2                | 3         | 0           | 0           | 15         |
| B       | SQL (GROUP BY)   | 2   | 2                | 2         | 0           | 0           | 16         |
| C       | Logical Design   | 4   | 2                | 1         | 1           | 0           | 7          |
| D       | Rel. Algebra     | 4   | 2                | 1         | 0           | 1           | 6          |
| E       | Rel. Algebra     | 4   | 2                | 1         | 1           | 0           | 5          |
| F       | Rel. Algebra     | 4   | 2                | 1         | 0           | 1           | 5          |
| F       | BCNF             | 3   | 2                | 2         | 0           | 0           | 15         |

The columns show the following data: “DPs” is the number of discrimination points used in the particular task, “Hamming Distance” is the minimal Hamming distance between two versions of the same task with respect to the number of discrimination points, “DPs wrong” is the number of discrimination points in the student’s answer that do not match the task description, “DPs adapted” is the minimal number of discrimination points that the student

found and adapted to match his/her assigned task description, “DPs omitted” is the number of discrimination points that were omitted in the student’s answer and last the “Group Size” column shows the number of students who worked on the same variant as the student with the cheating attempt but included discriminations points in the correct way.

Thus, six students were caught because they submitted solutions with wrong watermarks. Manual analysis revealed the original authors of the two SQL queries. One further student had a very similar solution to one that was known to be passed between students (he got the same exercise variant as the original author, therefore, one cannot say whether he would have detected the watermark). We did not try to search for the original authors of the non-SQL exercises. In these exercise types, there was less space for “personal style”. If we had used more values for the discrimination points, so that each discrimination vector is used only very few times (ideally once), that would have helped to identify the original authors. Five more students were caught with the log file analysis explained in Section 5. In total, 14 students were detected as cheating (out of 114, i.e. 12%).

As a test for false alarms, we applied semantic watermarks in the SQL tasks of a proctored exam (repetition exam with 19 participants in July 2022). Due to proctoring in a single room, cheating was quite unlikely. There were actually two queries with wrong watermarks, however, in each case, only one out of two discrimination points was wrong. In the first case, the student wrote “Informatik” instead of “Bioinformatik”. Since both are programs of study in our department, the discrimination point values were no secret in this case. In the second case, the student sorted by course title instead of the ID. Again, the columns are known to the student, and they might be intermixed. If a student might use a wrong discrimination value simply by mistake (without communication), the evidential value is small. However, two or more such wrong discrimination values would again be a strong evidence.

## 5 Verification by Log-File-Analysis

In the February 2022 exam, we worried that the students might already know the watermark method, because we applied it already in the exam of March 2021. Therefore, we added a different method: In all our exams, the students have the opportunity to test their queries with a given example database using a web-interface (Adminer). The same setup with a single “read only” database account for all students was used for the lab sessions and homeworks before. We told the students very clearly that (1) all accesses to our server during the exam are logged, and (2) the data will be analyzed for cheating attempts. However, it seems that many students did not foresee what actually can be done with the data. We matched the tried queries with the queries that were actually submitted in the exam. This gives the IP addresses used by many (not all) students. It might be that students share an IP address (e.g., if they live in the same student residence). However, if the same query  $Q$  (submitted by student  $S_1$ ) is tried from two different IP addresses  $I_1$  and  $I_2$ , and  $I_1$  is used for other queries submitted by  $S_1$ , and  $I_2$  is used for other queries submitted by  $S_2$ , it seems that  $S_1$  passed his query  $Q$  to  $S_2$ , and  $S_2$  tried it unchanged, and later modified it or decided to develop his/her

own solution. Of course, the query  $Q$  must be sufficiently complex so that it is extremely unlikely that the two students developed it independently.

We have no space to explain this method in detail, but it gives us a chance to verify the quality of the watermark method: (1) Are innocent students wrongly accused of cheating? (2) How many cheating students were not caught by the watermark method? The log analysis detected three clusters of students who exchanged SQL queries:

| Cluster ID | Number of Students | Students with wrong Watermark | Students      |
|------------|--------------------|-------------------------------|---------------|
| 1          | 3                  | 1                             | B, G, H       |
| 2          | 3                  | 1                             | D, I, J       |
| 3          | 5                  | 1                             | A, K, L, M, N |

In some cases, one can see in the log of an IP address, how the complex query  $Q$  suddenly appears and then is modified by changing the names of tuple variables, the case of SQL keywords, and so on. Even though it might be conceivable that  $S_1$  and  $S_2$  independently developed the same query  $Q$ , this process of obfuscation is a clear sign of guilt. In summary, the log analysis has hardened the case against the students who had the wrong watermark in their SQL queries, and made clearer who contributed to the cheating attempt by passing a solution.

## 6 Practical Advice

A common advice is “An ounce of prevention is worth a pound of cure” [Di03]. One certainly should make very clear before the exam that checks for cheating attempts will be done, and the consequences of being caught will be unpleasant. One must assume that more than a few students base the decision to cheat on a risk-benefit analysis. Everybody should understand that the risk is definitely not zero. It would also be unfair to set up a kind of “trap” for cheating students without informing them about such dangers. The goal must be that the students are more likely to overestimate the actual risks than to underestimate them.

In particular, it must clear that the professor really cares about academic integrity. The students must ask themselves whether they are able to outsmart the professor. If this paper is read by all of our students, they might be able to avoid being detected by these methods (although it would cost them at least some time). However, if a professor has once done things like the log analysis, there is the risk that other unforeseen methods are used next time.

Of course, one must respect data privacy laws and should contact the legal department of the university to be safe. One cannot tell the students the exact methods that will be used for detecting cheating attempts in their exam. Probably any algorithm for plagiarism detection can be avoided by the students as soon as they know it. It can be considered ethically

questionable when the used methods are not completely transparent. But situations, in which somebody decides to break the rules just because he can do it, often have no ethically completely satisfying solutions — precisely because the rules do not help anymore.

It is also important how one communicates the discovery of the cheating attempt to the students. We first tried to keep the details of our method secret (so that we might apply it again). That probably was a mistake. When a student gets such a message, he/she must decide whether to admit breaking the rules or not. After denying the cheating attempt, one cannot later change one's mind without also admitting that one lied before (which is very difficult). Therefore, the full evidence should be given to the student from the beginning. We also got the advice (from a non-lawyer) to ask the student whether he/she wants us to deliver the evidence. The answer “yes” might be understood as an agreement for data processing. For data privacy laws, it is also important to inform each student in a separate email, not containing the names of any other students, even if that would be an important part of the evidence. This might also have the advantage that students can individually decide whether to admit the cheating attempt.

One criticism of our work was also that we catch the “small criminals”, but so far we cannot do anything against really big cases of cheating — such as letting a paid expert write the entire exam. In future work, we plan to compare the programming style used in previous homeworks with the style in the exam (see also [MJ15, ORKS18]). It might be possible to do an additional oral exam (with full identification) in a few very suspicious cases. Of course, the possibility of such additional exams must be announced in advance.

## 7 Conclusions

Online exams seem to make cheating much simpler and less risky. However, due to the COVID-19 pandemic, they were often the only option. But even after the pandemic, new teaching methods and conveniences developed during the pandemic will remain: The expectations for online teaching are rising. If online exams could be done with trust in the integrity of the result, it would be possible to rethink the entire examination process. The techniques presented here might also be used for homeworks.

We used small variations in the exercises (nearly “invisible”) to prove that a solution was copied (because it contained the wrong “watermark”). For SQL queries, we verified the detected cheating attempts with data from the log of SQL queries tried during the exam. This also showed that only about half of the forbidden communication between the students was detected with the watermark technique. However, when the watermark technique shows a case of plagiarism, we know that the query from another student was really used for the query that was finally submitted (not only tried and thrown away). An interesting aspect of our work is that we wrote nearly the entire analysis in SQL (including recursive queries). More information is available from: [<https://users.informatik.uni-halle.de/~brass/sqlplag/>]. There will also be a longer version of this paper on [<https://arxiv.org/>].

## Bibliography

- [AHK03] Agrawal, Rakesh; Haas, Peter J.; Kiernan, Jerry: Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal*, 12:157–169, 2003.  
<https://doi.org/10.1007/s00778-003-0097-x>.
- [An22] Anonymous Authors: Betrug in Online-Klausuren (cheating in online exams), January 2022. <https://www.wiwi-treff.de/Lernen-and-Klausuren/Tauschungsversuch/Betrug-in-Online-Klausuren/Diskussion-88316>.
- [Ba17] Bawarith, Razan; Basuhail, Abdullah; Fattouh, Anas; Gamalel-Din, Shehab: E-exam Cheating Detection System. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 8(4):176–181, 2017.  
<https://dx.doi.org/10.14569/IJACSA.2017.080425>.
- [BG06] Brass, Stefan; Goldberg, Christian: Semantic errors in SQL queries: A quite complete list. *The Journal of Systems and Software*, 79(5):630–644, 2006.  
<https://doi.org/10.1016/j.jss.2005.06.028>,  
<https://dbs.informatik.uni-halle.de/sqlint/>.
- [Bi15] Binstein, Jake: On Knuckle Scanners and Cheating — How to Bypass Proctortrack, Examity, and the Rest. Blog, January 2015. <https://jakebinstein.com/blog/on-knuckle-scanners-and-cheating-how-to-bypass-proctortrack/>.
- [Cl20] Clark, Ted M.; Callam, Christopher S.; Paul, Noel M.; Stoltzfus, Matthew W.; Turner, Daniel: Testing in the Time of COVID-19: A Sudden Transition to Unproctored Online Exams. *Journal of Chemical Education*, 97(9):3413–3417, 2020.  
<https://pubs.acs.org/doi/pdf/10.1021/acs.jchemed.0c00546>.
- [DH05] Daly, Charlie; Horgan, Jane: A Technique for Detecting Plagiarism in Computer Code. *The Computer Journal*, 48(6):662–666, 2005. <https://doi.org/10.1093/comjnl/bxh139>,  
<https://www.researchgate.net/publication/31104286>.
- [Di03] Dick, Martin; Sheard, Judy; Bareiss, Cathy; Carter, Janet; Joyce, Donald; Harding, Trevor; Laxer, Cary: Addressing student cheating: definitions and solutions. *ACM SIGCSE Bulletin*, 35(2):172–184, June 2003. <https://doi.org/10.1145/782941.783000>.
- [DS17] D’Souza, Kelwyn A.; Siegfeldt, Denise V.: A Conceptual Framework for Detecting Cheating in Online and Take-Home Exams. *Decision Sciences Journal of Innovative Education*, 15(4):370–391, 2017. <https://doi.org/10.1111/dsji.12140>,  
<https://www.researchgate.net/publication/319967019>.
- [Eh21] Ehrlinger, Christina; Fritsch, Thomas; Fruth, Michael; Lehner, Franz; Scherzinger, Stefanie: Toolunterstützung für den Übungsbetrieb in der Datenbanklehre: Erfahrungen mit der Software Praktomat (Tool support for database labs: Experiences with the Software Praktomat). *Datenbank-Spektrum*, 21:91–98, 2021.  
<https://doi.org/10.1007/s13222-021-00374-y>.
- [FMG19] Foltýnek, Tomáš; Meuschke, Norman; Gipp, Bela: Academic Plagiarism Detection: A Systematic Literature Review. *ACM Computing Surveys*, 52(6), 2019.  
<https://doi.org/10.1145/3345317>.
- [GA11] Gross-Amblard, David: Query-preserving watermarking of relational databases and Xml documents. *ACM Transactions on Database Systems*, 36(1):1–24, 2011.  
<https://doi.org/10.1145/1929934.1929937>.

- [Ja21] Janke, Stefan; Rudert, Selma C.; Petersen, Änne; Fritz, Tanja M.; Daumiller, Martin: Cheating in the wake of COVID-19: How dangerous is ad-hoc online testing for academic integrity? *Computers and Education Open*, 2(100055), 2021. <https://doi.org/10.1016/j.caeo.2021.100055>.
- [KS19] Kleerekoper, Anthony; Schofield, Andrew: The False-Positive Rate of Automated Plagiarism Detection for SQL Assessments. In: UKICER: Proceedings of the 1st UK & Ireland Computing Education Research Conference. ACM, pp. 1–6 (Article No.: 6), 2019. <https://doi.org/10.1145/3351287.3351290>, <https://www.researchgate.net/publication/335480205>.
- [La08] Lafaye, Julien; Gross-Amblard, David; Constantin, Camelia; Guerrouani, Meryem: Watermill: An Optimized Fingerprinting System for Databases under Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 20:532–546, 2008. <https://doi.ieeecomputersociety.org/10.1109/TKDE.2007.190713>.
- [LC15] Lancaster, Thomas; Clarke, Robert: The implications of Plagiarism and Contract Cheating for the Assessment of Database Modules, 2015. <https://www.slideshare.net/ThomasLancaster/the-implications-of-plagiarism-and-contract-cheating-for-the-assessment-of-database-modules-teaching-learning-and-assessment-of-databases-2015>.
- [MAF21] Miedema, Daphne; Aivaloglou, Efthimia; Fletcher, George: Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In: Proceedings of the 17th ACM Conference on International Computing Education Research. ACM, pp. 355–367, 2021. <https://doi.org/10.1145/3446871.3469759>.
- [MJ08] McCart, James A.; Jarman, Jay: A Technological Tool to Detect Plagiarized Projects in Microsoft Access. *IEEE Transactions on Education*, 51(2):166–173, 2008. <https://ieeexplore.ieee.org/document/4455466>.
- [MJ15] Mirza, Olfat; Joy, Mike: Style Analysis for Source Code Plagiarism Detection. In: Plagiarism across Europe and Beyond — Conference Proceedings. European Network for Academic Integrity (ENAI), pp. 53–61, 2015. [https://academicintegrity.eu/conference/proceedings/2015/Mirza\\_Style.pdf](https://academicintegrity.eu/conference/proceedings/2015/Mirza_Style.pdf).
- [NJK19] Novak, Matija; Joy, Mike; Kermek, Dragutin: Source-code Similarity Detection and Detection Tools Used in Academia: A Systematic Review. *ACM Transactions on Computing Education*, 19(3):1–37, 2019. <https://doi.org/10.1145/3313290>.
- [NMA22] Noorbehbahani, Fakhroddin; Mohammadi, Azadeh; Aminazadeh, Mohammad: A systematic review of research on cheating in online exams from 2010 to 2021. *Education and Information Technologies*, 27:8413–8460, 2022. <https://doi.org/10.1007/s10639-022-10927-7>.
- [ORKS18] Opgen-Rhein, Julia; Küppers, Bastian; Schroeder, Ulrik: An Application to Discover Cheating in Digital Exams. In (Joy, Mike; Ithantola, Petri, eds): Proc. of the 18th Koli Calling International Conf. on Computing Education Research. ACM, pp. 1–5 (Article No.: 20), 2018. <https://doi.org/10.1145/3279720.3279740>.
- [PMP02] Prechelt, Lutz; Malpohl, Guido; Phlippsen, Michael: Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science*, 8(11):1016–1038, 2002. <https://doi.org/10.3217/jucs-008-11-1016>.

- [RC05] Russell, Gordon; Cumming, Andrew: Online Assessment and Checking of SQL: Detecting and Preventing Plagiarism. In: 3rd Workshop on Teaching, Learning and Assessment in Databases (TLAD), in 22nd British National Conference on Databases. HE Academy for Information and Computing Sciences, pp. 1–6 (Article No.: 6), 2005.  
<https://www.researchgate.net/publication/252140116>.
- [Si13] Singh, Anil: Detecting Plagiarism in MS Access Assignments. *Journal of Information Systems Education*, 24(3):177–180, 2013.  
<https://jise.org/Volume24/n3/JISEv24n3p177.html>.
- [SSS18] Scerbakov, Nikolai; Schukin, Alexander; Sabinin, Oleg: Plagiarism Detection in SQL Student Assignments. In (Auer, Michael E.; Guralnick, David; Simonics, Istvan, eds): *Teaching and Learning in a Digital World, Proc. of the 20th Conf. on Interactive Collaborative Learning (ICL)*. Springer, pp. 321–326, 2018.  
<https://www.researchgate.net/publication/319942726>.
- [TSV18] Taipalus, Toni; Siponen, Mikko; Vartiainen, Tero: Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education*, 18(3):1–29, 2018.  
<https://doi.org/10.1145/3231712>.



# Developing OERs for Teaching Database Systems<sup>1</sup>

## A Two-Year Effort of Four Universities of Applied Sciences

Thomas C. Rakow<sup>2</sup>, André Kless<sup>3</sup>, Charlotte Hasler<sup>4</sup>, Harm Knolle<sup>3</sup>, Heide Faeskorn-Woyke<sup>5</sup>, Inga Marina Saatz<sup>6</sup>, Jens Lambert<sup>4</sup>, Mareike Focken<sup>4</sup>

**Abstract:** In the project EILD.nrw, Open Educational Resources (OER) were developed for teaching database systems. Lecturers can use the tools and courses in a variety of learning scenarios. Students of computer science and application subjects can learn the complete lifecycle of databases. For this purpose, we developed and published quizzes, interactive tools, instructional videos, and courses for learning management systems, under a Creative Commons license. Find an overview of the developed OERs according to topic, description, teaching form, and format in this paper. Furthermore, we describe the implementation of licensing, sustainability, accessibility, contextualization, content description, and technical adaptability. The learning and teaching modules were used in ongoing classes and evaluated anonymously by students.

**Keywords:** database systems, open educational resources (OERs), higher education

## 1 Introduction

Using digital teaching and learning materials for teaching offers opportunities for design of new scenarios: *availability* anytime and anywhere, *customization*, especially with the individual speed of *reception*, automatable *review* of solutions, *traceability* of use as well as easy *reproducibility* [Ra21; RF19]. The content consists of presentations, scripts, and tasks. Activities can be forums and chats, surveys, and tests, enriched with gaming elements, all provided for learning management systems.

Overall, the EILD.nrw project developed about 80 open educational resources (OERs) for the life cycle of databases such as modelling, relational data model, SQL, and implementation

---

<sup>1</sup> Main parts of this work were supported by a grant of the Ministry of Culture and Science of the State of North Rhine-Westphalia as a cooperation project of the Digital University of North Rhine-Westphalia (DH.NRW).

<sup>2</sup> Hochschule Düsseldorf University of Applied Sciences, Faculty of Media, Düsseldorf, thomas.rakow@hs-duesseldorf.de,  <https://orcid.org/0000-0001-6374-9068>

<sup>3</sup> Hochschule Bonn-Rhein-Sieg University of Applied Sciences, Department of Computer Science, Sankt Augustin

<sup>4</sup> Hochschule Düsseldorf University of Applied Sciences, Faculty of Media, Düsseldorf

<sup>5</sup> Technische Hochschule Köln University of Applied Sciences, Faculty of Computer Science and Engineering Science, Gummersbach

<sup>6</sup> Fachhochschule Dortmund University of Applied Sciences and Arts, Faculty of Computer Science, Dortmund

of database management systems [EI22a; Ra22]. The project partners coming from four universities have many years of experience in teaching database systems with self-developed learning units. Every semester, about 1,000 students can acquire knowledge in supported self-study and work on practical tasks for the conception, creation, and programming of database systems.

The project was funded by the Ministry of Culture and Science (the Ministry) of the State of North Rhine-Westphalia (NRW) starting in the winter semester of 2020 as a cooperation project with the *Digitale Hochschule* of North Rhine-Westphalia (DH.NRW). The tools can be used for courses dedicated to studies in Informatics and Media, Business or Technical Informatics and applied to specific learning scenarios for blended learning. The content created in this as well as the other projects of the funding line OERContent.nrw are being made available under the CC BY-SA 4.0 license – sharing and editing with attribution and passing on under the same conditions [NN22]. Other included software and other material like logos are licensed as specified.

To our knowledge there is no recent initiative to develop such a huge amount of OERs for teaching database systems covering the main topics. We present the full scope of material and make it freely available to promote exchange in the database community and to improve teaching the database systems subject. Hence, some of the tools and courses developed in the project were presented at *LWDA'22* [Ra22].

We outline studies of OER usage of lecturers internationally, but mainly focus on German speaking countries' conditions of use for the database systems subject. We describe the requirements of OERs in general but also the requirements from the EILD.nrw project. The developed OER are listed respectively with their teaching form and their technical format. During development, we also focussed on quality assurance and compare our development with the findings published at project start [Ra21].

## 2 Related Work

The use and usability of OERs by lecturers were examined in many international studies with different focal points [BF21]. A study on how OERs must be designed so that lecturers can use them for the database systems subject is still pending. However, the results from existing, non-subject-specific studies already provide many clues that can be implemented for database systems related OERs. Numerous and frequently examined factors influence teachers in their decision to use an OER or not. Education, university policy and training can create the prerequisites for the use of OERs, such as such as knowledge and technical ability. [Pi16; WT17]. The nature of the OERs (content, technical, qualitative) including their metadata are crucial for findability and selection. There are solutions to improve metadata, ranging from cross-repository standardization [DD12] to subject-specific, ontology-based indexing (RJG14). Quality control and reusability are important for creation and provisioning. To ensure quality, one suggestion is to implement editorial processes [To19]. As a recent

initiative by lecturers, the *1<sup>st</sup> International Workshop on Data Systems Education* brought together case studies and experiences from the use of self-developed tools.

The usability of learning content and Open Educational Resources (OER) have been under discussion in Germany since 2000 [De16]. Slides, scripts, tasks, wikis, blogs, and video recordings are made available individually or as collections in online courses - the Massive Open Online Courses (MOOCs). Nevertheless, there is a great deal of helplessness as of how to integrate these offers into the curricula at German universities [De16].

At the 2008 autumn meeting in Düsseldorf, the German SIG Database Systems of the German Informatics Society (*GI-Fachgruppe Datenbanksysteme*) dealt with the topic "Quo Vadis: Forms of database training and further training"[Ra09]. A special issue of the journal *Datenbank-Spektrum* edited by the *GI-Fachgruppe Datenbanksysteme* was published introducing several digital and even one analog tool presented in the workshop [Ra09]. But it took a decade to address digital teaching again prominently in the community. At a workshop of the database systems conference BTW 2019 both procedures for teaching the database systems subject – digital communication, portals, blended learning – and their handling considered from the learner’s perspective were presented [RF19]. In 2021, two issues of *Datenbank-Spektrum* were dedicated to the digital lecture of database systems [STH21].

Our developments in the project EILD.nrw support the start done in 2019 by the German database community very well. We have extended our effort to make existing or newly developed tools available as OER. In our journal paper, we introduced the variables and their features for a good practical usage of OERs [Ra21]. The forms of licensing, sustainability, and accessibility, as well as the possibilities for contextualization and the content, the technical adaptability as well as compliance with data protection on platforms are the determining factors for exchanging digital content in teaching. In the project, we dedicated our efforts to make already existing or newly developed tools available as OER, and thereby fulfilling most of the requirements published.

### **3 Requirements for Database-related OERs**

Are there any peculiarities in applying digital teaching to the database systems subject? – The use of databases is taught using descriptive SQL programming. The desired results are specified, but not the path to these results. Usually, programming is taught first with procedural or object-oriented concepts [GI16]. Hence, the methods in SQL seem unfamiliar and therefore require a special approach on how loops or intermediate results can be "bypassed".

In computer science, digital tools are used professionally. Teaching has long been conducted using these tools – commercial ones, open source or self-developed. But does computer science in general not involve any practical work since “everything” is possible using

computers? – Not necessarily, even if originally analogue working methods such as pair programming can be mapped to digital formats. But in application areas such as media technology with the use of photo and film cameras, as well as the green room, or in media design using sheets of paper, analogue work continues [He17; Wo19]. Nevertheless, in the EILD project we focus on the digital tools which is enough work to do.

The requirements for the exchange of OERs [Ra21] are classified with EILD specific marks (Tab 1). Decisions must be made which types to choose resp. how to fulfil them. In the project EILD.nrw, some were already set by the by the Ministry’s call for proposals.

| Requirement                   | Types                      |                 |                         |                |
|-------------------------------|----------------------------|-----------------|-------------------------|----------------|
| <b>Licensing</b>              | private                    | without license | <u>commercial</u>       | <u>open</u>    |
| <b>Sustainability</b>         | permission to use          | customizability | technical feasibility   | <u>privacy</u> |
| <b>Accessibility</b>          | style                      | layout          | <u>language</u>         | language level |
| <b>Contextualization</b>      | <u>vocational training</u> | <u>school</u>   | <u>higher education</u> | professional   |
| <b>Content description</b>    | knowledge                  | learning goals  | competences             | learning paths |
| <b>Technical adaptability</b> | customization              | <u>platform</u> |                         |                |

Tab. 1: Requirements for the Exchange of OERs [Ra21], EILD-specific marks are underlined (must, optional)

- Licensing:** The licensing type was required to be Creative Commons with the variants *CC* for free at all, *CC-BY* with a mandatory credit for the original creation, and *CC-BY-SA* for the additional requirement using this licensing in case of further distribution for other versions [NN22]. In all variants, commercial use is allowed. We have chosen *CC-BY-SA* to be credited for our substantial work, and to encourage making changes – not only updates, as well as also language adaptations – available to the community.
- Sustainability:** Permission to use is granted by licensing and cannot be withdrawn. Customizability and technical feasibility as required by the request are achieved by using open software and/or state of the art products. As in modular systems, the resources should be selected based on metadata and a guide for the lecturers explaining their properties. Privacy must comply with the European General Data Protection Regulation (GDPR). The given distribution platform ORCA.nrw is operated by a German institution; thus, the responsibility is on their side first [OR22]. If the OERs log privacy-related data, their usage should be described and optionally made available.

- **Accessibility:** Due to different teaching concepts synchronous digital media like interactive presentations, courses, and (programmable) tools as well as asynchronous digital media like instructional videos/ screencasts and tests are offered. Due to their intended use in basic courses first, the initial content is in German. Translating the content into other languages is intended, but as there are no funds available in the project for translation, the OER community will have to rely on voluntary work. However, for example, learning instructions for textbooks in German - which are provided in our OER courses - , can only be mapped into other language settings with a great effort.
- **Contextualization:** The OERs are provided with descriptions, learning outcomes, and metadata. Teachers notes contain the didactic concept used in introductory videos and courses, explained in terms of best practices. The funding line's purpose was to encourage the use of digital materials for universities and target the higher educational sector. A professional level is not intended to be addressed. However, vocational education and training (VET) and secondary school courses may use some of the OERs provided.
- **Technical adaptability:** Easy customization is requested by the lecturers. It can be achieved in different ways according to the teaching form of an OER – video, tutorial, software tool, course – of an OER. The ministry requests the installation of ORCA.nrw as distribution platform [OR22]. Also, the OERs must be useable for the two learning management systems ILIAS and Moodle, which are widely used in NRW.

## 4 EILD Educational Resources

Thematically, for database systems, the following topics are recommended [Ke15, Ku15, Fa07]: *conceptual design, relational data model, SQL programming and mapping from models, database-based application development, and internals of database management systems (DBMS)*. Learning modules building on those fundamentals being suitable for master's degree programs are *distributed databases*, mainly *NoSQL databases*, and *data analysis*. The learning units developed are modular. Competencies (skills) to be taught are described in the recommendations of the German Informatics Society (GI) for Informatics [GI16] as well as for application subjects in Business Informatics [GI17], Media Informatics [He17; Wo19], and Technical Computer Science [GI18]. The study regulations and module descriptions of specific degree programs implement such recommendations institution related.

The tools and courses can be selected and adapted to the respective requirements of the lecturers and the type of their courses based on included guides. Using the open environments HTML, JavaScript, SQLite, and Jupyter notebooks, ensures to meet these requirements. For most of the tools, you can download texts in JSON format and change them to your

own needs. Software versioning via GitHub provisioning [EI22b], inclusion of didactic descriptions and evaluation results within the courses are also available.

The learning content is currently available on the ORCA.nrw platform [OR22] and partly as a repository on GitHub as well as an executable version on GitHub Pages. You can run an OER yourself on ILIAS, Moodle, or similar platforms by uploading the files provided on the ORCA.nrw platform or in GitHub as a packed file and unpacking it within the LMS. Another option is to generate the application from the source code using the build script. The content is thus freely available and easily accessible.

The self-assessment tests provide direct feedback on what was answered correctly or incorrectly. Thus, no user-specific data is processed at any time and no dependencies on external servers are implemented. Alternatively, you can use the versions provided via GitHub Pages making changes available immediately. Note, that the GitHub servers are outside the scope of the GDPR.

Table 2 shows an aggregation of OERs we have developed. In each case, the *topic*, a *description* of the content, the *form* of material, and the available *formats* are indicated.

| Topic  | Description  | Form                          | Format                           |
|--|--|-------------------------------|----------------------------------|
| <b>Overlapping Concepts</b>                    | 1300 questions with solution hints   | Multiple choice test          | JavaScript Pages<br>SQL Database |
|  | Generation of crossword puzzles from 200 items                               | Interactive tool              | JavaScript Pages                 |
|  | Tutorial for the three-layer architecture                                    | Instructional video           | Video with subtitles             |
| <b>Conceptual Design</b>                       | Modelling ER diagrams  | Instructional video           | Video with subtitles             |
|  |  | Interactive tool              | JavaScript Pages                 |
|  | Modelling UML class diagrams   | Interactive presentation      | H5P<br>SCORM                     |
| <b>Relational Data Model</b>                   | Introduction to the relational data model and the normalization of relations | Instructional videos          | Video with subtitles             |
|  |  | Interactive course            | Moodle Course                    |
|  |  | Interactive tool              | JavaScript Pages                 |
| <b>SQL Programming and Mapping from Models</b> | SQL querying and creating databases  | Interactive programmable tool | JavaScript Pages<br>SQLite DBMS  |
|  | Practicing triggers, functions, and procedures in PL/SQL                     | Interactive programmable tool | SQL DBMS - Oracle® application   |

Tab 2, part 1: Overview of the Developed OERs

| Topic                      | Description                                | Form                          | Format               |
|----------------------------|--|-------------------------------|----------------------|
|                            | Use of Oracle SQLcl® [La21]                | Instructional screencast      | Video with subtitles |
|                            | Use of Oracle SQL Developer® for PL/SQL    | Instructional screencast      | Video with subtitles |
|                            | Mapping the ER-Model or UML-classes to SQL | Interactive presentations     | JavaScript Pages     |
|                            | Optimising databases                       | Interactive presentation      | Video with subtitles |
| <b>Applications</b>        | Object-relational mapping                  | Instructional videos          | Video with subtitles |
|                            |  | Interactive presentation      | H5P                  |
|                            |  |                               | ILIAS module         |
| <b>Internals</b>           | The five-layer DBMS architecture           | Instructional video           | SCORM                |
|                            | Introduction to transaction management     | Instructional videos          | Video with subtitles |
|                            | Exercises to synchronize transactions      | Interactive tool              | JavaScript Pages     |
|                            | Creation of a B-tree index                 | Interactive tools             | JavaScript Pages     |
| <b>Distributed Systems</b> | Introduction to NoSQL databases            | Instructional videos          | Video with subtitles |
|                            | Distributed querying                       | Interactive course            | Video with subtitles |
|                            | MongoDB                                    | Interactive programmable tool | Jupyter Notebook     |
| <b>Data Analytics</b>      | Programming Apache SPARK with Python       | Interactive programmable tool | Jupyter Notebook     |
|                            | Text analyses                              | Interactive programmable tool | Jupyter Notebook     |
|                            | Image analyses                             | Interactive programmable tool | Jupyter Notebook     |

Tab 2, part 2: Overview of the Developed OERs

## 5 Quality Assurance

All new and further developed tools were presented and discussed in regular meetings including all project members such as professors, scientific and student staff. Demonstrations take place once per quarter. After the presentation of the innovations, the used contents and concepts are discussed professionally for improvement and review. The discussion was complemented by the exchange about didactic concept, technical implementation, usability, and the need for actual use in teaching - with the associated difficulties.

The learning and teaching modules were used in ongoing classes and evaluated anonymously by students [RF21]. A mixture of quantitative and qualitative surveys has proven to be helpful. The evaluation results were documented in the learning units to provide students and lecturers with an aid for selection.

While the quantitative information only records the valuation by the students, specific suggestions and reasons for the valuation could be given in the free text. The free-text option was well received by the students, and the suggestions made for improvement could be evaluated in combination with the scaled answers and integrated into the modules accordingly. For example, the "*Podcast: Introduction to SQLcl from Oracle*" was rated as helpful according to quantitative questioning. In addition, criticism of the visibility of individual elements in some web browsers was also expressed in the free text. The screencast was therefore revised according to accessibility guidelines and the problem was solved with improved browser compatibility.

Students also wanted more helpful tips beyond the DBMS's error messages. For this purpose, all faulty SQL statements were logged and compared with the stored sample solution [FBS20]. Suggestions from student were not only related to the course content itself, but also to its integration into teaching [RF21]. There was also a desire to improve the instructions for using the tools, as well as suggestions for structuring the course.

In most cases, the peer discussions as well as student feedback have led to specific improvements in OER and will be continued.

## 6 Conclusions

With our developments in the project, we could verify our findings published at the beginning of the project in *Datenbank-Spektrum* [Ra21]: (1) *a mix of synchronous and asynchronous digital media*, (2) *a sustainable licensing*, (3) *a clear didactic concept*, (4) *a low-threshold student participation* (through reporting, voting, anonymous questionnaires, suggestion boxes, breakout, and question sessions, etc.), (5) *the agreement among the lecturers* of a degree program to limit the variety of tools and methods, and (6) *the use of interactive tools*.

We believe, there is not just one concept for successful teaching, but rather an interaction of several influencing factors will decide on the success. The EILD.nrw project has made a significant contribution to these findings.

Focusing on teaching as well as on the successful learning process allows a different view on the integration of OERs, which in this scenario serves to improve teaching [BF21]. More purposeful than the question of how teachers use or can use OER is the following question: How can OERs enhance teaching as well as the learning process? And what do teachers need for this?

In many projects we have developed predecessors of these OERs. We would like to thank the students who helped us to improve the content through their questions in lectures and practical courses. The students of our degree programs in the semesters starting in the winter semester 2020 up to the summer semester 2022 took thankfully part in the quality assurance surveys. Especially, we would like to thank the students who made the results of their student research work available to us: *Anne Giesen, Arabella Jackszis, Frederic Cieslik, Jonas Baur and Monika Jousen*. Together with the authors of this article, *Alexander Kosmehl, Björn Salgert, Christian Schindler and Melanie Beutel* worked on the EILD.nrw project. Our heartfelt thanks go to them all.

## References

- [AFM22] Aivaloglou, E.; Fletcher, G.; Miedema, D.: DataEd'22 - 1st International Workshop on Data Systems Education: Bridging Education Practice with Education Research. In: Proceedings of the 2022 International Conference on Management of Data. SIGMOD '22, Association for Computing Machinery, Philadelphia, PA, USA, pp. 2556–2557, 2022, ISBN: 9781450392495, URL: <https://doi.org/10.1145/3514221.3524079>.
- [BF21] Baur, J.; Focken, M.: OER-Nutzung durch Lehrende. Arbeitspapiere des Lehrgebiets Datenbanken und E-Business 2021/3, pp. 32–42, 2021, URL: <https://doi.org/10.20385/opus4-3372>.
- [DD12] Dichev, C.; Dicheva, D.: Open Educational Resources in Computer Science Teaching. In: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education. SIGCSE '12, Association for Computing Machinery, Raleigh, North Carolina, USA, pp. 619–624, 2012, ISBN: 9781450310987, URL: <https://doi.org/10.1145/2157136.2157314>.
- [De16] Deutscher Bildungsserver: Machbarkeitsstudie zum Aufbau und Betrieb von OER-Infrastrukturen in der Bildung (Stand: Februar 2016). 2016.
- [EI22a] EILD.nrw: GitHub repository, 2022, URL: <https://github.com/EILD-nrw/>, visited on: 08/01/2023.
- [EI22b] EILD.nrw: Website. 2022, URL: <https://eild.nrw>, visited on: 08/01/2023.

- [Fa07] Faeskorn-Woyke, H.; Bertelsmeier, B.; Riemer, P.; Bauer, E.: Datenbanksysteme: Theorie und Praxis mit SQL2003, Oracle und MySQL. Pearson Deutschland GmbH, 2007.
- [FBS20] Faeskorn-Woyke, H.; Bertelsmeier, B.; Strohschein, J.: A decision tree approach for the classification of mistakes of students learning SQL, a case study about SELECT statements. In: DELFI 2020–Die 18. Fachtagung Bildungstechnologien der Gesellschaft für Informatik eV. Gesellschaft für Informatik eV, pp. 211–216, 2020, URL: <https://dl.gi.de/handle/20.500.12116/34162>.
- [He17] Heinecke, A. M.; Kindsmüller, M. C.; Noss, C.; Rakow, T. C.; Rumpfer, M.; Wolters, C.: Medieninformatik 2017: Berufsbilder, Färbungen, Curricula und Erfahrungen. In: Mensch und Computer - Workshopband. Gesellschaft für Informatik eV, pp. 459–464, 2017, URL: <https://doi.org/10.18420/muc2017-ws10-0426>.
- [Ju17] Jung, R.: Rahmenempfehlung für die Ausbildung in Wirtschaftsinformatik an Hochschulen (März 2017). Gesellschaft für Informatik eV, 2017.
- [KE15] Kemper, A.; Eickler, A.: Datenbanksysteme. 10. Auflage, De Gruyter, Oldenbourg, 2015.
- [Ku15] Kudraß, T.: Taschenbuch Datenbanken. Carl Hanser Verlag GmbH Co KG, 2015.
- [La21] Lambert, J.: Podcast: Einführung in Oracle SQL Command Line (SQLcl)./, Hochschule Düsseldorf, 2021, URL: <https://doi.org/10.20385/opus4-3461>.
- [Ma18] Maehle, E.: Curriculum für Bachelor-und Masterstudiengänge Technische Informatik (März 2018). Gesellschaft für Informatik eV, 2018.
- [NN22] N.N.: Creative commons licenses. 2022, URL: <https://creativecommons.org/licenses/>, visited on: 08/01/2023.
- [OR22] ORCA.nrw: Studium und Lehre digital unterstützt. 2022, URL: <https://orca.nrw>, visited on: 08/01/2023.
- [Pi16] Piedra, N.; Chicaiza, J.; López, J.; Caro, E. T.: Integrating OER in the design of educational material: Blended learning and linked-open-educational-resources-data approach. In: 2016 IEEE Global Engineering Education Conference (EDUCON). IEEE, pp. 1179–1187, 2016, URL: <https://doi.org/10.1109/EDUCON.2016.7474706>.
- [Ra09] Rakow, T. C.; Faeskorn-Woyke, H.; Schiefer, B.; Vossen, G.; Wäsch, J.: Tools für die Lehre im Fach Datenbanken. Datenbank-Spektrum 9/29, pp. 5–13, 2009, URL: [https://www.researchgate.net/publication/220102832\\_Tools\\_fur\\_die\\_Lehre\\_im\\_Fach\\_Datenbanken](https://www.researchgate.net/publication/220102832_Tools_fur_die_Lehre_im_Fach_Datenbanken).
- [Ra21] Rakow, T. C.; Faeskorn-Woyke, H.; Saatz, I. M.; Knolle, H.: Es EILD–Anforderungen an die Publikation freier Lerneinheiten (OER) im Fach Datenbanken. Datenbank-Spektrum 21/2, pp. 111–120, 2021, URL: <https://doi.org/10.1007/s13222-021-00373-z>.

- [Ra22] Rakow, T. C.; Faeskorn-Woyke, H.; Saatz, I. M.; Knolle, H.: OER Tools and Courses for Teaching Database Systems as Developed in the Project EILD.nrw (Extended Abstract). In: LWDA'22. To be published in: CEUR Workshop Proceedings, 2022.
- [RF19] Digitale Lehre im Fach Datenbanken, Gesellschaft für Informatik eV, 2019, pp. 97–98, URL: <https://doi.org/10.18420/btw2019-ws-09>.
- [RF22] Rakow, T. C.; Focken, M.: EILD.nrw–Die Evaluation von Lehrinhalten im Fach Datenbanken. In: Forschungsreport 2021. Hochschule Düsseldorf, pp. 138–139, 2022, URL: <https://nbn-resolving.org/urn:nbn:de:hbz:due62-opus-36892>.
- [RJG14] Ruiz-Iniesta, A.; Jiménez-Díaz, G.; Gómez-Albarrán, M.: A semantically enriched context-aware OER recommendation strategy and its application to a computer science OER repository. IEEE Transactions on Education 57/4, pp. 255–260, 2014, URL: <https://doi.org/10.1109/TE.2014.2309554>.
- [STH21] Scherzinger, S.; Thor, A.; Härder, T.: Themenhefte "Digitale Lehre im Fachgebiet Datenbanksysteme". Datenbank-Spektrum 21/1/2, Mar. 2021, URL: <https://link.springer.com/journal/13222/volumes-and-issues/21-1/https://link.springer.com/journal/13222/volumes-and-issues/21-2>.
- [To19] Towey, D.; Reisman, S.; Chan, H.; Demartini, C.; Tovar, E.; Margaria, T.: OER: Six perspectives on global misconceptions and challenges. In: 2019 IEEE International Conference on Engineering, Technology and Education (TALE). IEEE, pp. 1–7, 2019, URL: <https://doi.org/10.1109/TALE48000.2019.9225943>.
- [Wo19] Wolters, C.; Kindsmüller, M. C.; Heinecke, A. M.; Rakow, T. C.; Dahm, M.; Jent, S.; Rumpler, M.: Medieninformatik 2019: Kompetenzorientierte Lehr-Lernszenarien in der Medieninformatik. In: Mensch und Computer 2019 - Workshopband. Gesellschaft für Informatik eV, pp. 512–517, 2019, URL: <https://dx.doi.org/10.18420/muc2019-ws-305>.
- [WT17] Wang, T.; Towey, D.: Open educational resource (OER) adoption in higher education: Challenges and strategies. In: 2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE). IEEE, pp. 317–319, 2017, URL: <https://doi.org/10.1109/TALE.2017.8252355>.
- [Zu16] Zukunft, O.: Empfehlungen für Bachelor-und Masterprogramme im Studienfach Informatik an Hochschulen (Juli 2016). Gesellschaft für Informatik eV, 2016.



# Enhancing Explainability and Scrutability of Recommender Systems

Azin Ghazimatin<sup>1</sup>

**Abstract:** Our increasing reliance on complex algorithms for recommendations calls for models and methods for explainable, scrutable, and trustworthy AI. While explainability is required for understanding the relationships between model inputs and outputs, a scrutable system allows us to modify its behavior as desired. These properties help bridge the gap between our expectations as end users and the algorithm's behavior and accordingly boost our trust in AI. Aiming to cope with information overload, recommender systems play a crucial role in filtering content (such as products, news, songs, and movies) and shaping a personalized experience for their users. Consequently, there has been a growing demand from the information consumers to receive proper explanations for their personalized recommendations. To this end, we put forward proposals for explaining recommendations to the end users. These explanations aim at helping users understand why certain items are recommended to them and how their previous inputs to the system relate to the generation of such recommendations. Such explanations usually contain valuable clues as to how a system perceives user preferences and more importantly how its behavior can be modified. Therefore, as a natural next step, we develop a framework for leveraging user feedback on explanations to improve their future recommendations. We evaluate all the proposed models and methods with real user studies and demonstrate their benefits at achieving explainability and scrutability in recommender systems.

**Keywords:** Recommender Systems; Explainable AI; Scrutability

## 1 Introduction

Our increasing reliance on complex algorithms for recommendations calls for models and methods for explainable and scrutable AI. While explainability helps us understand the cause of a decision made by an algorithm [Mi19], a scrutable system enables users to correct system's assumptions when needed [TM07]. These properties bring about trust by bridging the gap between humans and AI.

Aiming to personalize content based on user preferences, recommender systems are perceived as advice-givers that can improve our acceptance through explanations [RRS15]. With the emergence of more complex models [KBV09] outperforming the simpler and more explainable ones [Sa01], *Explainable AI* has progressively received more attention from the Recommender Systems (RecSys) community [ZC20]. Lack of transparency in recommender systems can have a direct impact on user acceptance, as based on the content personalized

---

<sup>1</sup> Saarland University and Max Planck Institute for Informatics, 66123 Saarbrücken, Germany aghazima@mpi-inf.mpg.de

for users, they may feel that the system is labeling them inappropriately<sup>2</sup> or misusing their private information<sup>3</sup>. To highlight the gravity of this matter, recently, laws have been passed to establish users' right to explanations [GF17].

Despite the close tie between explainability and scrutability [BR20], they do not necessarily entail each other. In other words, knowing why the algorithm makes particular choices may not be sufficient for realizing how to modify it. For instance, imagine a user of an online movie streaming service who is frequently recommended with action movies. The system explains its choices by drawing connections between the recommended movies and the action movies the user previously watched on the platform. Now, consider a situation where the user wants to stop receiving such movies as they do not entirely match her interest. Here, the provided explanations do not act as a precise guide as to how they can effectively exert control over their recommendations. Therefore, scrutability in recommender systems requires separate consideration and handling.

The following sections delve into the concepts of explainability and scrutability and describe our contributions towards realizing these objectives.

## 2 Explainable Recommendations

Recommender systems aim at delivering personalized content such as products, movies, books, and songs to their users. The chosen content is often visualized in a ranked list, where the order reflects the relevance of the items to the user. To compute these relevance scores, recommender systems usually train models based on various inputs collected from their users. User inputs can be explicit (e.g., rating or liking an item) or implicit (e.g., watching a movie or listening to a song). The abundance of implicit signals has facilitated data collection by service providers.

Providing the systems with an enormous amount of data over time, users might not be able to remember all the details of their interactions, and hence experience difficulty in understanding why they receive certain items as their recommendations. This problem particularly worsens when users do not even have access to the complete history of their interaction with the system, a phenomenon referred to as *inverse privacy* [GW16]. Therefore, it is imperative for the recommender systems to be explainable, i.e., to enable users to understand the relationships between their own input to the system and the recommendations they receive.

To illustrate how a recommendation can be explained, imagine a user who is a member of a social cataloging website like Goodreads<sup>4</sup> and receives a book recommendation, titled *Recovery: Freedom from Our Addictions*. Example 2.1 presents a possible way of

---

<sup>2</sup> <https://www.wsj.com/articles/SB1038261936872356908>

<sup>3</sup> <https://www.wired.co.uk/article/tiktok-filter-bubbles>

<sup>4</sup> <https://www.goodreads.com>

explaining this recommendation to the user by outlining a connection between the given recommendation and their past actions on the platform:

**Example 2.1** You  $\xrightarrow{\text{liked}}$  *Becoming*  $\xrightarrow{\text{has genre}}$  *Autobiography*  $\xrightarrow{\text{belongs to}}$  *Recovery: Freedom from Our Addictions*

In Section 2.1, we describe a framework for generating such explanatory paths based on user's interactions with a given platform.

Apart from describing *why* a certain item is relevant to a user, recommender systems are also expected to be able to explain the rankings, i.e., to reason *why* a certain item is *more relevant* than the others. For instance, the following statement explains the cause of receiving the book *Recovery: Freedom from Our Addictions* as the *top-ranked* recommendation:

**Example 2.2** You are recommended with the book *Recovery: Freedom from Our Addictions* because you liked the books ***Becoming*** and ***Dreams from My Father***. *If you did not like these two books, your top-ranked recommendation would be the book ***Food and Nutrition***.*

Example 2.2 shows that *liking* the books *Becoming* and *Dreams from My Father* is the key reason that the book *Recovery: Freedom from Our Addictions* is more relevant to the user than the book *Food and Nutrition*. The blue text in this example demonstrates the causality between user's previous action and system's outcome. Such explanations are referred to as counterfactual; they pinpoint those user actions whose absence would result in a different recommendation for them. Identifying the true reasons behind the recommendations, these explanations pave the way towards scrutability, i.e., they help shed light on how users can control what they see as their recommendations. In Section 2.2, we describe a method for generating counterfactual explanations.

## 2.1 Post-hoc Explanations for Black-Box Recommenders

Web users interact with a huge volume of content every day, be it for news, entertainment, or inside social conversations. To save time and effort, users are progressively depending on curated *feeds* for such content. A feed is a stream of *individualized* content items that a service provider tailors to a user. One example of a feed is the list of questions and answers recommended to users on Quora<sup>5</sup>. Since a feed is a one-stop source for information, it is important that users understand *how items in their feed relate to their profile and activity on the platform*.

To help users understand these relationships, we introduce FAIRY, a **F**ramework for **A**ctivity-**I**tem **R**elationship discover**Y**. FAIRY enables users to discover useful and surprising

<sup>5</sup> <https://www.quora.com>

relationships between their own actions on the platform and their recommendations. For this, we first model the user's local neighborhood on the platform as an interaction graph. This graph is constructed solely from the information available to the user. In a user's interaction graph, the set of simple paths connecting the user to her feed item are treated as pertinent explanations. Example 2.1 illustrates one such explanatory path. Next, FAIRY scores the discovered explanations with learning-to-rank models built upon users' judgements on relevance and surprisal of the explanation paths. Longitudinal user studies on two social platforms, *Quora* and *Last.fm*<sup>6</sup>, demonstrate the practical viability and user benefits of this framework in different domains. For more detailed analysis, refer to [Gh21a; GSW19].

## 2.2 Counterfactual Explanations for Recommendations

FAIRY's explanations are post-hoc, i.e., they are decoupled from the underlying recommender system. While essential for enhancing transparency of black-box models, these explanations are not actionable; they may mislead the user when used for modifying the system's behavior. To overcome this limitation, we introduce PRINCE, a method for **Provider-side Interpretability with Counterfactual Evidence**.

PRINCE enables graph-based recommenders with personalized PageRank at their core [NK19] to generate concise and counterfactual explanations for their users. To see an example of such explanations, see Example 2.2. PRINCE explains the most relevant recommendation to the user by identifying the minimum number of their previous actions whose removal from the user history could displace the top-ranked item. To find the minimal counterfactual explanations from an exponential search space, PRINCE uses a polynomial-time algorithm, and hence it is efficient.

Experiments on two real-world datasets show that PRINCE provides more compact explanations than intuitive baselines. Insights from a crowdsourced user-study demonstrate the viability of such action-based explanations. For further details refer to [ghaz; Gh20].

## 3 Scrutable recommendations

A *scrutable* recommender system allows its users to tell the system when it is wrong and enables users to steer their recommendations accordingly [TM07]. This feature is particularly useful when users experience drifts in their interests or when the system cannot correctly infer their preferences. Evidence suggests that scrutability can improve user's engagement level and their satisfaction [HKN12; Kn12; PB15].

Critique-enabled recommenders have already taken the first step towards scrutability. These systems employ a feedback mechanism called *critiquing* that enable users to express their

---

<sup>6</sup> <https://www.last.fm>

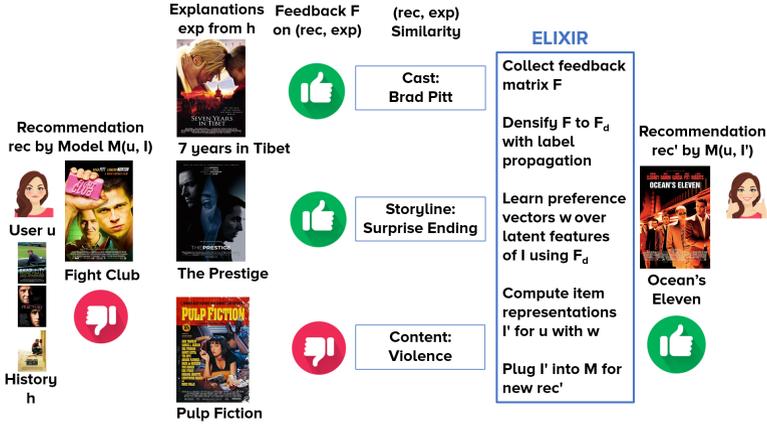


Figure 1: Example illustrating the intuitions behind ELIXIR.

dissatisfaction with some characteristics of the recommended item [CP12]. For instance, imagine a student who relies on an online service like Yelp <sup>7</sup> to find a nice place to have dinner. The recommended restaurants, however, are not suitable for her as they are mostly expensive and far from her place. In this scenario, she will benefit from system-suggested critiques such as *show me a cheaper* or *closer restaurant* that enables her to explore other options that suit her interest better. In the next section, we describe how recommender systems can leverage user feedback on explanations as a critiquing mechanism to improve their future recommendations.

### 3.1 Using Explanations to Improve Recommender Systems

Explanations contain valuable information on *why* a certain item is recommended to the user. We posit that the similarity between the recommended item and its corresponding explanation speaks to the reason behind receiving the recommendation in the first place. This drives the design of a feedback collection mechanism to learn users' fine-grained preferences. Fig. 1 shows an illustrative scenario. User  $u$  receives a recommendation for the movie *Fight Club* ( $rec$ ) based on her online history and factors like item-item similarities. This is accompanied by an explanation referring to three items, all previously liked by  $u$  and being similar, in some aspects, to  $rec$ . We have  $exp_1$ : *Seven Years in Tibet*,  $exp_2$ : *The Prestige*, and  $exp_3$ : *Pulp Fiction*. The system generated these three items for explanation because:

- $exp_1$  features the actor Brad Pitt who also stars in  $rec$ ,
- $exp_2$  has a surprise ending, similar to  $rec$ ,

<sup>7</sup> <https://www.yelp.com>

- $exp_3$  contains violent content, like *rec*.

Now suppose that user  $u$  loves Brad Pitt and surprise endings but hates disturbing violence (she likes *Pulp Fiction* for other reasons like its star cast and dark comedy, that dominated her opinion, despite the violence). When receiving *rec* with the above explanation, user  $u$  could give different kinds of feedback. The established way is to simply dislike *rec*, as a signal from which future recommendations can learn. However, this would completely miss the opportunity of learning from how user  $u$  views the three explanation items. The best feedback would be if user  $u$  could inform the system that she likes Brad Pitt and surprise endings but dislikes violence, for example, by checking item properties or filling in a form or questionnaire. However, this would be a tedious effort that few users would engage in. Also, the system would have to come up with a very fine-grained feature space of properties, way beyond the usual categories of, say, movie genres.

To facilitate efficient critiquing, we introduce ELIXIR, a framework for (**E**fficient **L**earning from **I**tem-based **e**Xplanations **I**n **R**ecommenders). ELIXIR enables recommenders to obtain user feedback on pairs of recommendation and explanation items, where users are asked to give a binary rating on the shared aspects of the items in a pair. To incorporate the collected feedback, we propose a method to learn user-specific latent preference vectors used for updating item-item similarities. The underlying intuition is to increase (decrease) the distance of disliked (liked) items and the like to the user's profile, such that the quality of future recommendations is improved. Our framework is instantiated using generalized graph recommendation based on personalized PageRank. Insightful experiments with a real user study show significant improvements for movie and book recommendations over item-level feedback. For a detailed analysis, refer to [Gh21a; Gh21b].

## 4 Conclusion

In this work, we studied explainability and scrutability of recommender systems. We introduced FAIRY, a framework for generating post-hoc explanations for black-box recommenders. We further proposed PRINCE, a provider-side interpretability tool, to provide users with concise and counterfactual explanations. Putting explanations into action, we lastly introduced ELIXIR, a framework for leveraging user feedback on explanations to improve their future recommendations. Our studies demonstrate the benefits of explanations for both end users and service-providers: users gain insight into the personalization process, and service providers enhance their users' experiences by offering more transparency and facilitating user control through feedback on explanations. We hope that this work sparks interest in the community towards responsible systems and pushes forward the mindsets and infrastructures required for trustworthy AI.

## Literatur

- [BR20] Balog, K.; Radlinski, F.: Measuring Recommendation Explanation Quality: The Conflicting Goals of Explanations. In: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25–30, 2020. S. 329–338, 2020, URL: <https://doi.org/10.1145/3397271.3401032>.
- [CP12] Chen, L.; Pu, P.: Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction* 22/1, S. 125–150, 2012.
- [GF17] Goodman, B.; Flaxman, S. R.: European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Mag.* 38/3, S. 50–57, 2017, URL: <https://doi.org/10.1609/aimag.v38i3.2741>.
- [Gh20] Ghazimatin, A.; Balalau, O.; Saha Roy, R.; Weikum, G.: PRINCE: Provider-side Interpretability with Counterfactual Explanations in Recommender Systems. In: WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020. S. 196–204, 2020, URL: <https://doi.org/10.1145/3336191.3371824>.
- [Gh21a] Ghazimatin, A.: Enhancing explainability and scrutability of recommender systems, Diss., Saarland University, Saarbrücken, Germany, 2021, URL: <https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/32590>.
- [Gh21b] Ghazimatin, A.; Pramanik, S.; Roy, R. S.; Weikum, G.: ELIXIR: Learning from User Feedback on Explanations to Improve Recommender Models. In: WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021. S. 3850–3860, 2021, URL: <https://doi.org/10.1145/3442381.3449848>.
- [GSW19] Ghazimatin, A.; Saha Roy, R.; Weikum, G.: FAIRY: A Framework for Understanding Relationships Between Users' Actions and their Social Feeds. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019. S. 240–248, 2019, URL: <https://doi.org/10.1145/3289600.3290990>.
- [GW16] Gurevich, Y.; Wing, J. M.: Inverse privacy. *Commun. ACM* 59/7, S. 38–42, 2016, URL: <https://doi.org/10.1145/2838730>.
- [HKN12] Hijikata, Y.; Kai, Y.; Nishida, S.: The relation between user intervention and user satisfaction for information recommendation. In: Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012. ACM, S. 2002–2007, 2012, URL: <https://doi.org/10.1145/2245276.2232109>.
- [KBV09] Koren, Y.; Bell, R. M.; Volinsky, C.: Matrix Factorization Techniques for Recommender Systems. *Computer* 42/8, S. 30–37, 2009, URL: <https://doi.org/10.1109/MC.2009.263>.

- [Kn12] Knijnenburg, B. P.; Bostandjiev, S.; O'Donovan, J.; Kobsa, A.: Inspectability and control in social recommenders. In: Sixth ACM Conference on Recommender Systems, RecSys '12, Dublin, Ireland, September 9-13, 2012. S. 43–50, 2012, URL: <https://doi.org/10.1145/2365952.2365966>.
- [Mi19] Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.* 267/, S. 1–38, 2019, URL: <https://doi.org/10.1016/j.artint.2018.07.007>.
- [NK19] Nikolakopoulos, A. N.; Karypis, G.: RecWalk: Nearly Uncoupled Random Walks for Top-N Recommendation. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019. S. 150–158, 2019, URL: <https://doi.org/10.1145/3289600.3291016>.
- [PB15] Parra, D.; Brusilovsky, P.: User-controllable personalization: A case study with SetFusion. *Int. J. Hum. Comput. Stud.* 78/, S. 43–67, 2015, URL: <https://doi.org/10.1016/j.ijhcs.2015.01.007>.
- [RRS15] Ricci, F.; Rokach, L.; Shapira, B., Hrsg.: Recommender Systems Handbook. Springer, 2015, ISBN: 978-1-4899-7636-9.
- [Sa01] Sarwar, B. M.; Karypis, G.; Konstan, J. A.; Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001. S. 285–295, 2001, URL: <https://doi.org/10.1145/371920.372071>.
- [TM07] Tintarev, N.; Masthoff, J.: A Survey of Explanations in Recommender Systems. In: Proceedings of the 23rd International Conference on Data Engineering Workshops, ICDEW 2007, 15-20 April 2007, Istanbul, Turkey. S. 801–810, 2007, URL: <https://doi.org/10.1109/ICDEW.2007.4401070>.
- [ZC20] Zhang, Y.; Chen, X.: Explainable Recommendation: A Survey and New Perspectives. *Found. Trends Inf. Retr.* 14/1, S. 1–101, 2020, URL: <https://doi.org/10.1561/15000000066>.

# Adaptive Architectures for Robust Data Management Systems

Tiemo Bang<sup>1</sup>

**Abstract:** “Form follows function” is a well-known expression by the architect Sullivan asserting that the architecture of a building should follow its function. “Adaptive Architectures for Robust Data Management Systems” is a dissertation asserting that DBMS architectures should follow changing workload and hardware to robustly achieve high DBMS performance. The dissertation first evaluates how workload and hardware affect the performance of DBMSs with static architectures. This evaluation concludes that static DBMS architectures degrade DBMS performance under changing workload and hardware, and hence the DBMS architecture has to become adaptive. Subsequently, adaptation concepts for the architecture of single-server and multi-server DBMSs are proposed. These concepts focus fine-grained adaptation of DBMS architectures and are realized through asynchronous programming models. These programming models decouple the implementation of DBMS components from fine-grained architectural optimization. Thereby, optimizers can derive novel architectures better fitting individual DBMS components, leading to high and robust DBMS performance under changing conditions.

**Keywords:** Database Management System; Architecture; Adaptation; Scale-Up; Scale-Out

## Summary of “Adaptive Architectures for Robust Data Management Systems”

*“Unceasingly the essence of things is taking shape in the matter of things. [. . . ] It is the pervading law of all things organic and inorganic, of all things physical and metaphysical, [. . . ] that form ever follows function. This is the law. Shall we, then, daily violate this law in our art?”* — Sullivan, 1896 in *The Tall Office Building Artistically Considered*

Sullivan asserts that the architecture of a building should follow from its function. The dissertation “Adaptive Architectures for Robust Data Management Systems” [Ba22a] asserts the same for the architecture of a database management system (DBMS). It particularly asserts that changing workload and hardware for DBMSs necessitate changing (adaptive) DBMS architectures. The dissertation contributes to adaptive DBMS architectures through publications on the performance of DBMSs with static architectures [Ba22b, Ba20a] and concepts for the adaptation of DBMS architectures for single-server DBMSs [Ba20b] and multi-server DBMSs [Ba21]. The following provides a brief summary of the overall work.

---

<sup>1</sup> University of California Berkeley, EECS, 1860 Le Roy Ave., CA 94709 Berkeley, United States of America  
tbang@berkeley.edu

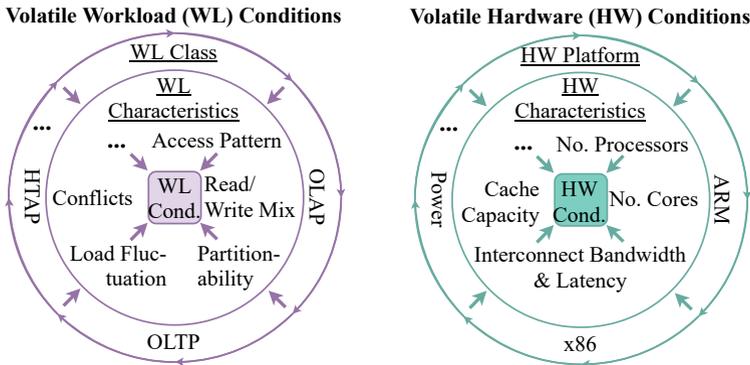


Fig. 1: Overview of the volatile conditions challenging DBMS performance. DBMSs are exposed to a range of workload classes and hardware platforms. These influence the displayed workload and hardware characteristics and lead to challenging volatile conditions under which DBMSs still need to perform well. The DBMS design has to align with these changing characteristics, otherwise performance can degrade significantly.

**DBMSs are challenged by changing workloads and hardware.** Today's DBMSs handle various types of workloads for online commerce, banking, and fraud detection. These workloads differ in key characteristics like the read-write pattern and further fluctuate depending on the popularity of individual data items. Similarly, modern DBMSs are typically compatible with a wide range of hardware platforms which still have different characteristics like different types or numbers of processors. This presents DBMSs with the challenge to perform well despite workload and hardware characteristics that widely vary, possibly at runtime.

The performance evaluation of this dissertation, along with a review of related work, identifies complex effects of various workload and hardware characteristics on the performance of DBMSs with static architectures, cf. [Ba22b]. As summarized in Figure 1, the different workload classes and hardware platforms lead to a wide range of characteristics that make for volatile workload and hardware conditions a DBMS design has to cope with. The evaluation surfaces a complex interaction between these characteristics with which the DBMS components and the surrounding DBMS architecture have to align precisely, otherwise performance can degrade sharply. Accordingly, the conclusion is that robust performance under changing conditions requires precise adaptation of the entire DBMS design, including the DBMS architecture.

**DBMSs remain tethered to their static architecture.** Serving changing workloads and supporting diverse hardware platforms is non-trivial for DBMSs. The design of the DBMS components and the DBMS architecture have to fit the conditions. DBMS components, like the query execution engine, indeed have developed adaptation mechanisms for that reason.

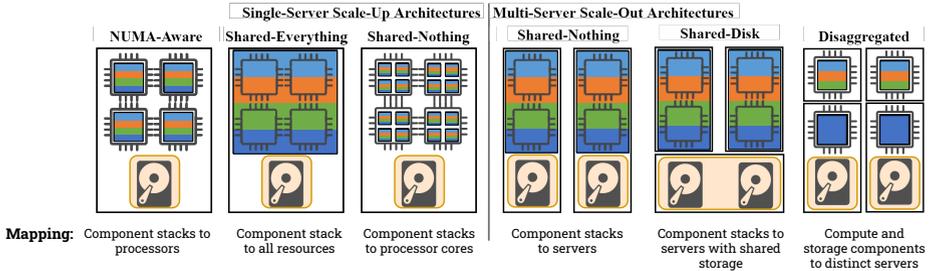


Fig. 2: Overview of the established DBMS architectures for single-server and multi-server DBMS with static deployment strategies, indicated by the colored boxes. Each architecture follows a static deployment strategy that determines a resource partitioning (shown by the sizes of the boxes) and deployment of DBMS components (shown by the colors the boxes). Most architectures uniformly deploy a stack of each DBMS component onto the resource partitions.

Mechanisms like just-in-time query compilation, for example, allow the query execution engine to flexibly specialize its operators to different workload or hardware. However, such adaptation is limited to the internals of these DBMS components. There is no adaptation outside of or across DBMS components, as the surrounding DBMS architecture is inflexible.

Today DBMS designers select from a handful of static architectures with rigid deployment strategies. As shown in Figure 2, these static architectures predetermine a fixed resource partitioning. The DBMS components are then deployed onto these resource partitions, as stacks containing an instance of each component. For example, the NUMA-aware architecture accounts for multi-processor hardware through resource partitions per processor onto which the DBMS components are deployed. Such tailoring, however, is a static design decision. This decision at design-time leads to an architecture baked into the DBMS implementation that only fits predetermined workload and hardware. Moreover, this manual decision also causes oversimplification. That is, all the static architectures uniformly deploy DBMS components onto coarse-grained resource partitions ignoring the unique workload and hardware effects within individual DBMS components. As a result, current static DBMS architectures are not flexible and lack sophistication, such that these architectures become unfit and degrade DBMS performance.

### Adaptive DBMS Architectures

The overall idea for the adaptation of DBMS architectures is to flexibly compose fine-grained “building blocks” of the DBMS to a best-fit architecture. This dissertation emphasizes fine-grained adaptation of the architecture to suit the individual internal functions of DBMS components. Besides a relief from the re-implementation effort, the goal is to create a navigable optimization space for DBMS architectures, enabling optimizers to flexibly mimic any existing architecture and more importantly to derive entirely new architectures.

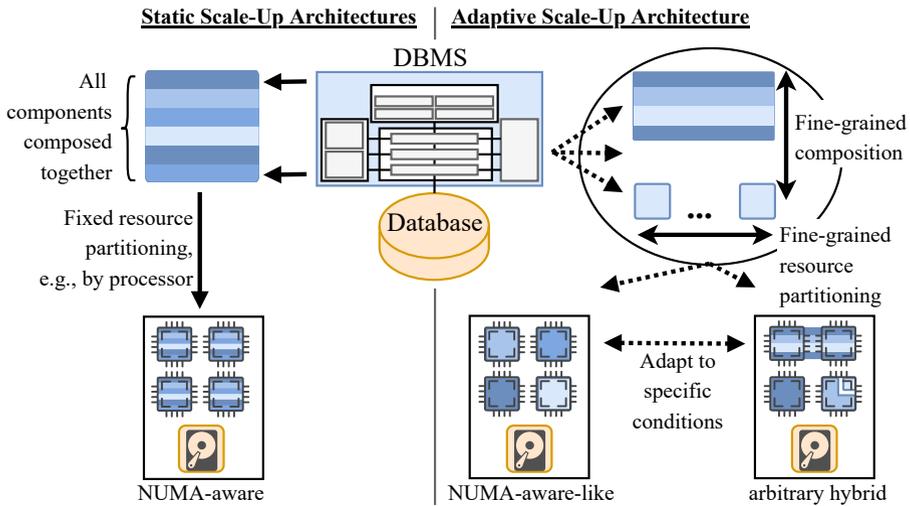


Fig. 3: Designing static scale-up architectures (left) versus proposed fine-grained adaptation (right). Left: Current architectures are statically designed with static deployment strategies for specific hardware/workload and treat all DBMS components uniformly, e.g., the NUMA-aware architecture dictates the resources partitioning by processor which each contain the complete stack of components. Right: The proposed adaptive scale-up architecture enables fine-grained arbitrary resource partitioning for arbitrary compositions of DBMS components, facilitating best-fit architectures.

**Adaptive Single-Server DBMS Architecture.** The adaptive single-server architecture introduces flexible configuration for DBMSs on shared-memory hardware, cf. [Ba20b]. It addresses the distinct adaptation demands on a single server, orthogonal to adaptation across multiple servers.

The key factors for single-server DBMS architectures to address are shared-memory data structures and the constraint to fixed resources. Single-server DBMS architectures must carefully organize these limited resources, especially in regard to efficient concurrent execution on shared data structures. For example, the static NUMA-aware architecture employs processor-sized resource partitions to enforce processor-local concurrent execution of DBMS components and explicit coordination of DBMS components across processors. Generally, single-server architectures strive for a resource partitioning that effectively balances partition-local concurrent execution on shared data structures within DBMS components and the explicit coordination of components across resources partitions. Static DBMS architectures strike this balance only for specific predetermined workloads and hardware, but likely become unfit under changing conditions. Instead, the proposed adaptive architecture enables flexible adaptation at the granularity of individual data structures of DBMS components, for best-fit single-server architectures across changing workloads and hardware.

The proposed adaptive single-server architecture introduces abstractions for the generic implementation of DBMS components and the fine-grained configuration of the architecture. It introduces a programming model of asynchronous data-aware tasks for DBMS components. These data-aware tasks divide DBMS components into units of execution on individual data structures, enabling the fine-grained configuration of the DBMS architecture. As shown in Figure 3, this configuration enables optimizers to flexibly configure arbitrary architectures with heterogeneous resource partitions tailored to individual data structures (or composites thereof). In contrast to the static “prepackaged” architectures, optimizers thereby can form novel architectures best-fit for each data structure under changing workload and hardware. The benefits of this flexible configuration are demonstrated through an optimizer for automatic throughput maximization.

**Adaptive Multi-Server DBMS Architecture.** The adaptive multi-server architecture offers flexible orchestration of the DBMS across elastic network-connected resources, cf. [Ba21]. It complements the above adaptation within a single server.

The key factors for multi-server architectures to address are the network communication between resources and the flexible addition/removal (scaling) of resources. Especially in the cloud, multi-server DBMSs can utilize an elastic resource pool to maintain high performance when facing heterogeneous and fluctuating workloads. Yet, at the same time, the network communication across this resource pool also poses a significant challenge. Multi-server architectures therefore aim to balance resource load and network communication overhead when orchestrating the DBMS across this resource pool. Static multi-server architectures, however, strike this balance only for specific workloads, e.g., the shared-nothing architecture for partitionable workloads. Changing or mixed workloads and the properties of individual DBMS components are not well reflected. For example, analytical queries and the query executor component can generally utilize more resources than transactional queries and the transaction manager components.

The adaptive multi-server architecture introduces individually optimized architectures for simultaneous queries. Rather than compromising on a common architecture, it simultaneously orchestrates query-optimized architectures across elastic resources. It defines a reactive programming model with generic DBMS actors for that reason. These actors are flexibly instrumented to temporarily enact (parts of) DBMS components and together enact a DBMS architecture. The execution and data flow of the DBMS are essentially broken down into streams of events and data items that can be routed across the resource pool. As illustrated in Figure 4, the routing and interleaving of these event and data streams therefore enables the simultaneous orchestration of multiple query-optimized architectures. The dissertation focuses on exploring the new degrees of freedom offered by this adaptive multi-server architecture.

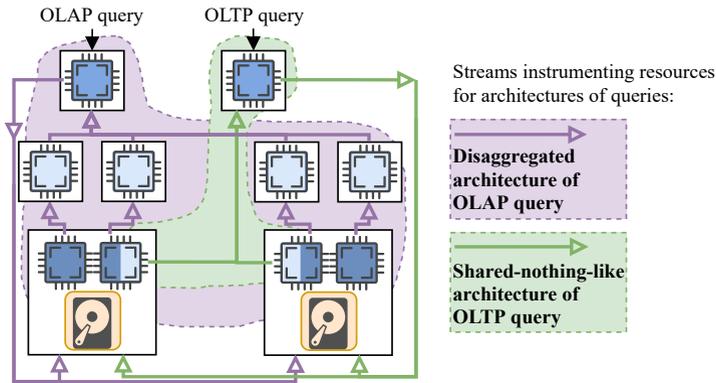


Fig. 4: The adaptive multi-server DBMS architecture for simultaneous optimization to distinct queries. The adaptive architecture instruments DBMS actors on elastic resources via event and data streams to enact DBMS components, allowing to orchestrate query-optimized architectures for the concurrent OLTP and OLAP queries. The purple streams orchestrate a disaggregated architecture for the OLAP query, while the green streams simultaneously orchestrate a shared-nothing architecture for the OLTP query.

## Conclusion

The key findings are that both the realized adaptive single-server and multi-server architectures prove effective and efficient. Under changing transactional and mixed workloads, the proposed adaptive architectures generally perform at least on par with the individually best state-of-the-art architecture. Indeed, when adopting novel better-fit architectures, all existing architectures are outperformed, e.g., when partitioning resource at a granularity unlike any of the existing architectures or when distinctly orchestrating architectures for queries of mixed workloads. Overall, the proposed flexible and precise adaptation demonstrates higher and more robust performance.

While the findings exhibit novel better-fit architectures only for a subset of possible workload and hardware conditions, this dissertation overall indicates high potential for adapting architectures with the proposed concepts. As the proposed concepts make a vast optimization space generally navigable, optimizers will be able to adapt DBMS architectures flexibly and more precisely to many workloads and hardware. Instead of fragile static architectures, the proposed adaptive architectures thus provide a necessary element for DBMSs to achieve high and robust performance under changing workload and hardware.

**Acknowledgments** I would like to thank the many people who provided advise, companionship, and sponsoring. Special thanks go to Carsten Binnig whose steadfast guidance led to this dissertation.

## Bibliography

- [Ba20a] Bang, Tiemo; May, Norman; Petrov, Ilia; Binnig, Carsten: The Tale of 1000 Cores: An Evaluation of Concurrency Control on Real(ly) Large Multi-Socket Hardware. In: Proceedings of the 16th International Workshop on Data Management on New Hardware. ACM, 2020.
- [Ba20b] Bang, Tiemo; Oukid, Ismail; May, Norman; Petrov, Ilia; Binnig, Carsten: Robust Performance of Main Memory Data Structures by Configuration. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20),. ACM, New York, NY, USA, 2020.
- [Ba21] Bang, Tiemo; May, Norman; Petrov, Ilia; Binnig, Carsten: AnyDB: An Architecture-less DBMS for Any Workload. In: 11th Annual Conference on Innovative Data Systems Research (CIDR '21). 2021.
- [Ba22a] Bang, Tiemo: Adaptive Architectures for Robust Database Management Systems. PhD thesis, Technische Universität, Darmstadt, 2022.
- [Ba22b] Bang, Tiemo; May, Norman; Petrov, Ilia; Binnig, Carsten: The Full Story of 1000 Cores: An Examination of Concurrency Control on Real(ly) Large Multi-Socket Hardware. In: The VLDB Journal. 2022.



Demo Track



# JumpXClass: Explainable AI for Jump Classification in Trampoline Sports

Lucas Woltmann,<sup>1</sup> Katja Ferger,<sup>2</sup> Claudio Hartmann,<sup>1</sup> Wolfgang Lehner<sup>1</sup>

**Abstract:** Movement patterns in trampoline gymnastics have become faster and more complex with the increase in the athletes' capabilities. This makes the assessment of jump type, pose, and quality during training or competitions by humans very difficult or even impossible. To counteract this development, data-driven solutions are thought to be a solution to improve training. In recent work, sensor measurements and machine learning is used to predict jumps automatically and give feedback to the athletes and trainers. However, machine learning models, and especially neural networks, are black boxes most of the time. Therefore, the athletes and trainers cannot gain any insights about the jump from the machine learning-based jump classification. To better understand the jump execution during training, we propose JumpXClass: a tool for automatic machine learning-based jump classification with explainable artificial intelligence. Using elements of explainable artificial intelligence can improve the training experience for athletes and trainers. This work will demonstrate a live system capable to classify and explain jumps from trampoline athletes.

**Keywords:** machine learning; applied AI; explainable AI; sports; trampoline

## 1 Introduction

Over the years, professional sports have experienced a boost in athletic capabilities. Athletes are able to reach new heights of performance in their respective areas. This has led to the need for better training tools including digital ones to better capture the athletes' performance. As a part of gymnastics, trampoline sport has experienced the same developments. Here, one central point is the capture of jumps via near-body sensors and their automatic classification to support the athletes and trainers via auxiliary digital tools [He11, Ca18, Wo22b]. The idea encompassed by all publications is that athletes can improve their performance and training experience by quantifying the exercise through near-body sensor measurements.

In recent research, Woltmann et al. promote using ML in a feedback system for trampoline athletes to improve training experience [Wo22a]. The work enables athletes and trainers to visualize the sensor measurements and automatically classify their jumps using a deep feed-forward neural network (NN). All shown data can be manipulated interactively. Part of the presented work in this demo is based on this preliminary work.

---

<sup>1</sup> Technische Universität Dresden, Dresden Database Research Group, Dresden, Germany, [firstname.lastname@tu-dresden.de](mailto:firstname.lastname@tu-dresden.de)

<sup>2</sup> Justus Liebig University Giessen, Institute of Sport Science, Giessen, Germany, [katja.ferger@sport.uni-giessen.de](mailto:katja.ferger@sport.uni-giessen.de)

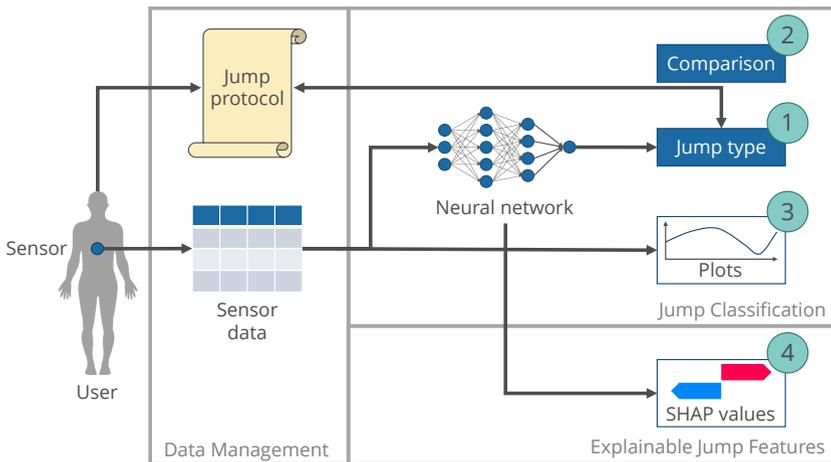


Fig. 1: System overview of JumpXClass. The numbers represent the different use cases.

The high volume of time series data and their black box characteristic are major drawbacks of data-driven ML-based approaches. Decisions, like the jump classification, are generally not traceable by or explainable to the user. This makes it challenging to analyze the interaction between data and model for the subsequent discussions of model quality and application. The scientific field researching this phenomenon and its solutions is called eXplainable Artificial Intelligence (XAI). In this area, ML black boxes are analyzed to conclude about the inner decision-making. For NNs, feature influences are one part of this analysis [LL17, KL17]. These influences quantify the input of an NN according to its influence on the decision. One representative are *shapley additive explanations* (SHAP) values [LL17]. SHAP values are a game theoretic approach modeling the exclusion of features and, therefore, their influence on the model's output. With SHAP, any ML model input feature is assigned a numerical value, representing the absolute influence on the decision. For the jump classification NN, these values represent the probability that the input feature adds to or subtracts from all possible classes.

In this demonstrator, we combine the approaches of a feedback system, automated ML-based classification, and XAI to build an ML-driven exercise assessment tool for trampoline athletes and trainers. During the live demonstration, we will show that combining these concepts brings several advantages to athletes and trainers by incorporating digital technologies into the training.

## 2 Demonstrator Features

In this section, we present the features and modules of our demonstrator JumpXClass. Three core concepts are implemented in three equivalent modules called *Data Management* to handle user data and meta data, *Jump Classification* to make the ML components accessible, and *Explainable Jump Features* to include the XAI parts. The three general modules of the demonstrator are presented in Figure 1 indicated by gray frames. Each module will add up on the following and improve the feedback given to the athletes and trainers. The four use cases, indicated by numbers, are detailed in Section 3.

### 2.1 Data Management

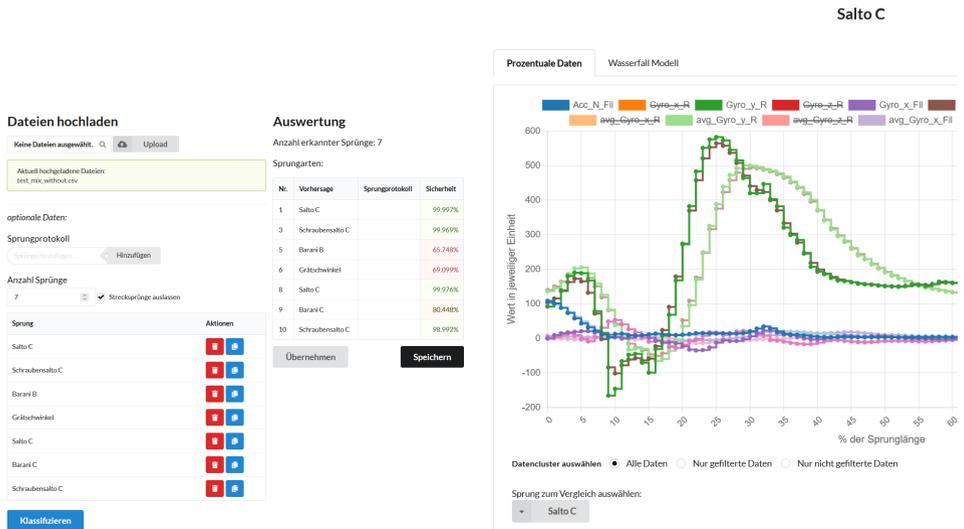
The tool works on the sensor data collected by an accelerometer and a gyroscopic sensor on the back of the athlete. After exporting the data from the sensor, the athletes upload their exercise data, usually containing ten consecutive jumps, into the tool. These are usually CSV files exported from the sensors with a measurement resolution of 500 Hz. Additionally, athletes need to document their exercise in a jump protocol manually. Our tool allows for the creation of such a protocol. Therefore, the athletes can store both their data and meta data for further use. This is important for the reproducibility of a training session or competition and the self-assessment of the athletes. The combined sensor data and jump protocols can also be used as labeled training data within the tool.

### 2.2 Jump Classification

Another aspect of this work is the direct use of a model that takes the sensor data and classifies all jumps according to their jump type. There are 148 jump types that our model needs to distinguish. The model is trained and tested on data containing around 2,500 jumps. The data was labeled by athletes (and their trainers) competing on a national level.

The classification gives direct feedback to the athletes about their high-level performance, i.e., if they performed the jump correctly. This can be useful for training by giving the athletes hints on the jumps they need to improve their performance. Another capability is the direct checking of the jump protocol. Misabeled or mixed-up entries in the protocol can be spotted more quickly if the NN contradicts a protocol entry.

All sensor channels are visualized as time series plots, as shown in Figure 2b. Athletes and trainers can analyze individual jumps from the jump data in more detail. The sensor data contains fine-grained acceleration and orientation information. This enables the analysis of wrong movements during individual jumps. As an additional feature, a comparison to previous jumps or an averaged jump for a jump type is possible via an inlay plot as presented by the lines with a lighter color in Figure 2b. This provides a comparative movement analysis



(a) Use Cases (1) and (2): The jump protocol (left) and the automatic jump classification (right). (b) Use Case (3): The time series plot for a specific jump including a reference jump (lighter colors).

Fig. 2: Example screens for three use cases.

to spot improvements or errors. Specifically, trainers can have a more detailed look at the jump performance and give quantitative and qualitative feedback to the athlete.

### 2.3 Explainable Jump Features

The jump classification gives a high-level assessment of a jump but no detailed feedback about the ML model’s decision. With XAI, the ML model can give athletes and trainers feedback. To achieve this, we analyze the jump classification NN with SHAP and report the SHAP values to the user via a waterfall plot. With the SHAP influences for each gyroscopic measurement, the athlete can see what movement patterns define a specific jump type according to the NN. For the first time in trampoline sports, XAI helps to identify the part of a jump most influential for good execution. This opens up new possibilities to support training and enhance athletes’ performances with ML and XAI. Additionally, the trainers can identify the measurements leading the NN to a wrong classification and give feedback to the athlete about a changed movement pattern for the jump. The athletes can get manifold feedback from the ML and XAI components to improve their training.

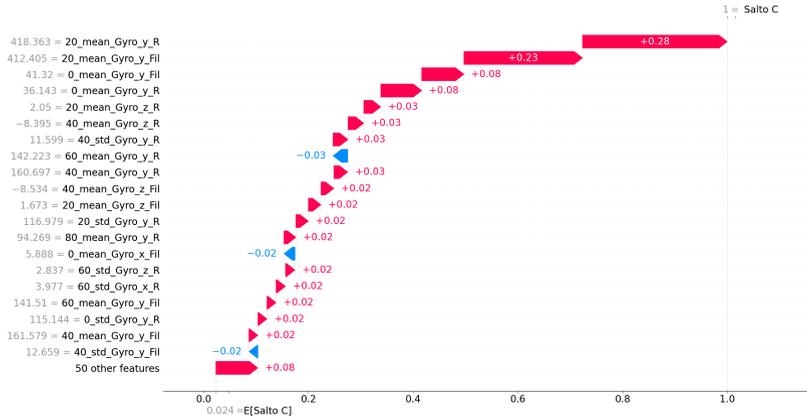


Fig. 3: Use Case (4): The SHAP waterfall plot for a specific jump.

### 3 Demo Description

In the live demo session, the users will see four use cases: (1) the competition mode, (2) the jump protocol, (3) the measurement plots, and (4) the XAI jump features. The cases are depicted and highlighted according to their number in Figure 1. Every use case has a dedicated target group. Whereas use cases (2), (3), and (4) are meant for athletes and trainers, use case (1) is aimed at competition judges. In (1), only the NN’s jump classification from an exercise data set is shown. Therefore, the judges can assess the exercise’s difficulty score.

In use case (2), the athletes upload their data for an exercise. The gyroscopic sensor produces time series data sets containing all required measurement channels. Additionally, the athletes can add their jump protocols and store them as meta data for the sensor data. Use case (1) is a subpart of use case (2) since it uses the same sensor data. Both use cases (1) and (2) are presented in Figure 2a. After uploading the data and adding the protocol, the NN automatically analyzes and classifies the jumps into jump types according to [Wo22b]. Every classification is annotated by a percentage detailing the confidence of the NN for this classification. Lower confidence scores usually show that a jump was not cleanly executed. Another point is the direct comparison of the automatic classification and the jump protocol. Color coding shows the users any discrepancies between these two parts and allows for a high-level analysis of the athletes’ performance.

Use case (3) plots the uploaded sensor measurement channels as time series plots by clicking on a specific jump in the jump protocol from the first use case, as shown in Figure 2b. The user can (de-)select each channel separately to be plotted or not. Another feature is the comparison view. Here, the athletes or trainers can choose either another jump from the jump protocol or an averaged representative for the specific jump type to be plotted for comparison with the original jump. This gives allows the users to compare their performance to a quasi-standard.

Use case (4) shows the SHAP value waterfall plot for the same selected jump as in use case (3). As detailed in Figure 3, the plot provides feedback regarding the classification and according to which features the NN decided on the jump type. The plot helps scientific experts to assess the jump quality from a technical perspective. The measurements and their influence on the decision can be analyzed and used to find the most important movement of a jump or a deviation of the athlete from a high-quality jump. Additionally, the SHAP values can be used to debug the data and model, either to re-record the jumps or to fine-tune the NN. A re-recording is necessary if a certain feature greatly influences a wrong classification and the corresponding movement is not part of the jump. Here, the athlete would execute the jump again and generate new sensor data for verification. If a feature influences the wrong classification but its corresponding movement is part of the jump, the NN needs to be adjusted through hyperparameter tuning or retraining with new data.

## 4 Conclusion

In this work, we have shown JumpXClass, a feedback system for trampoline athletes and trainers based on ML models and XAI. The multitude and interplay of features make the tool a valuable asset for athlete quality assessment. It can help to identify errors in jumps or quality metrics of jump types. We argue that using JumpXClass can also highlight the advantages of ML and (X)AI in sports and other applications. For future work, we plan to verify these advantages with actual trampoline gymnasts to obtain feedback for the tool.

## Bibliography

- [Ca18] Camomilla, Valentina; Bergamini, Elena; Fantozzi, Silvia; Vannozzi, Giuseppe: Trends supporting the in-field use of wearable inertial sensors for sport performance evaluation: A systematic review. *Sensors*, 18(3):873, 2018.
- [He11] Helten, Thomas; Brock, Heike; Müller, Meinard; Seidel, Hans-Peter: Classification of trampoline jumps using inertial sensors. *Sports Engineering*, 14(2):155–164, 2011.
- [KL17] Koh, Pang Wei; Liang, Percy: Understanding black-box predictions via influence functions. In: *International conference on machine learning*. PMLR, pp. 1885–1894, 2017.
- [LL17] Lundberg, Scott M; Lee, Su-In: A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.
- [Wo22a] Woltmann, Lucas; Ferger, Katja; Hartmann, Claudio; Lehner, Wolfgang: Exercise assessment in trampoline sport by automated jump classification. In: *Tagungsband zum 14. Symposium der Sektion Sportinformatik und Sporttechnologie der Deutschen Vereinigung für Sportwissenschaft*. p. 23, 2022.
- [Wo22b] Woltmann, Lucas; Hartmann, Claudio; Lehner, Wolfgang; Rausch, Paul; Ferger, Katja: Sensor-based Jump Detection and Classification with Machine Learning in Trampoline Gymnastics. *German Journal of Exercise and Sport Research*, 12 2022.

# UniDash: Interactive Dashboard for Data Driven Insights on Universities

Mirjam Bayer<sup>1</sup> <sup>2</sup>; Yorik Timo Hansen<sup>1</sup>; Kimberley Kosbü<sup>1</sup>; Andrea Kulow<sup>1</sup>; Peer Kröger<sup>1</sup>

**Abstract:** Universities, like other institutions or companies, are under steady quality control in pursuit of improvement. Generating pertinent insights from consistently collected data at universities is one of the many possibilities to integrate data science into our educational system. The proposed prototype dashboard is designed for educational institutions to visually assess their shifts in relevant topics such as diversity, accessibility, and planning aspects. This paper shows the workflow and dashboard using the UnivIS database of Kiel University for extracting and preprocessing the data. The proposed demo revealed interesting insights, such as how, in the planning stage, lecture halls are selected with only 50% capacity utilization; rooms for fewer than 50 people are planned to be used at 100% capacity. The demonstration web application can be tested in German at [unidash.tk](http://unidash.tk).

**Keywords:** interactive dashboard; academic advising; data-driven decision-making

## 1 Introduction

In the past decade, it became of paramount interest to assess and improve the quality at universities, leading up to tackling the required changes as described by Baker and Lenhardt in [BL08]. In order to evaluate development, one must be able to measure and trace numbers describing the feature of interest. However, collecting data is only the first step to extracting valuable insights. Data scientists are trained to compare and visualize the generated data. To perform quality evaluation on complex questionnaires, and working groups are required, which are laborious, and demanding of time and monetary resources. Department heads of large institutions like universities often lack the time to set up these complex evaluation tools for their institution. Therefore, we approached the challenge and developed a dashboard showcasing the progression of a university using its own data collected over the past 20 years. We developed a workflow for extracting the data from open-source tools and implemented a dashboard, making the shifts in the data clearly traceable. The derivable insights range from the capacity utilization of classrooms to the percentage of accessible classrooms used by each department.

Data mining is increasingly utilized for meta-evaluation in the academic world, for example, the assessment of the student body [Th22], [Gu20], or optimizing teaching methods [MJ12].

---

<sup>1</sup> Kiel University, Department of Computer Science, Christian-Albrechts-Platz 4, 24118 Kiel, {miba,stu227160,stu207503,stu229089,skr}@informatik.uni-kiel.de

<sup>2</sup> Parts of this work has been done in the MARISPACE-X project funded by German Ministry of Economy (BMWi)

The untapped potential of using academic data in the education system is further analyzed in [We12] and [Fi20], concluding that incorporating data mining will enhance informed decision-making, leading to optimized teaching effectiveness. This demonstrates the growing significance of mining the available data in educational systems. In this paper, we propose a workflow and prototype dashboard to visualize patterns and trends in data from Kiel University. The main contributions can be summarized as follows:

- Workflow for the extraction of data from a university management system
- Dashboard with interactive visualizations

## 2 UnivIS Data

Large institutions like universities have to solve complex planning problems every semester. The solution must combine human resources, room vacancies, and examination regulations to name just a few of the constraints all timetables must fulfil. To combine all the required information, multiple database services are used in the German network of universities. Like the universities of, e.g., Erlangen-Nuremberg, Bamberg, and Lübeck, Kiel University uses UnivIS [STU99] to store the relevant data for the planning of the semesters. UnivIS is a relational database, with an API to extract data as XML. The database schema is adapted for the university process, to seamlessly combine teaching, room and human data. Through multiple API requests, [Ki22b], we were able to pull all of the collected data. We merged the XML responses into a local SQL database using the same scheme as the UnivIS database. In combination with UnivIS, Kiel University’s Department of Computer Science uses a module database (the ModulDB) to store additional details regarding their courses. Data from both sources were gathered for the dashboard.

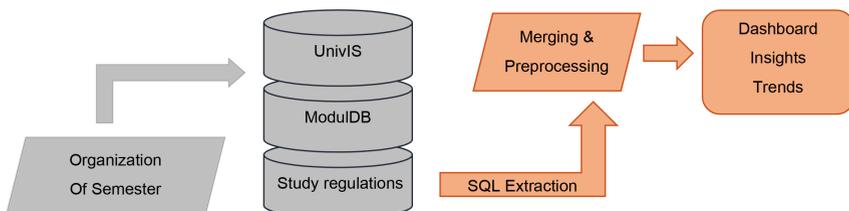


Fig. 1: Workflow of generating the dashboard, grey fields: implemented processes at universities, orange: our implemented contribution

## 3 Dashboard

The implemented dashboard is designed for universities to showcase their strengths and weaknesses in a customized open-source application. It targets universities to enable them to

get a clear view of their own position on a variety of current and relevant aspects. Because the application uses existing data sets the dashboard can also reveal past developments. The proposed workflow is shown schematically in Figure 1. Steps that have already been taken at the universities are drawn in grey. The orange steps entail our contribution built upon the available databases.

On the dashboard, there are currently nine topics visualized. Each topic is designated to a research question in a timely fashion. In this paper, we explain the following four questions in more detail.

- Are the rooms used according to their capacity?
- How wheelchair accessible are the courses of different faculties? Is wheelchair accessibility evenly distributed across faculties?
- Can students reach their classes on time, or are locations too far apart? (Exemplary for two subjects of study)
- Kiel: 'Best Prof' Award voted by students. What insights can be extracted from past winners?

## 4 Technical details

UnivIS Kiel provides a public API, [Ki22b], supplying the required data for this dashboard. As a result of the database being maintained and data being inserted manually, one has to presume many errors regarding spelling, coherence, and missing values. This was especially problematic when working with the addresses for classrooms. From abbreviations for buildings, misspelled street names, and missing zip codes to lacking street numbers, all possible errors had to be caught. Using regular expressions, almost all addresses could be extracted, but this required expert knowledge of the university community, mainly concerning common abbreviations.

Regarding the dashboard deployment, the Python library *plotly* [P115] was used for easy and fast implementation. This also allows for simple modification and transfer to other universities with few requirements of HTML and CSS knowledge. The code, including the SQL requests for extracting from UnivIS, is available on a GitHub repository.<sup>3</sup>

## 5 Demonstration

Using the example of Kiel University, the dashboard can be viewed in German at `uni.dash.tk`. The elaborated topics are presented in separate tabs, where one or more interactive graphs visualize the corresponding data for the different departments and semesters. The navigational toolbar on the left leads to the different topic pages.

---

<sup>3</sup> <https://github.com/doubleblind44/unidash>

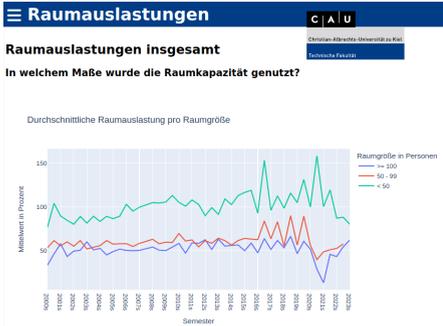


Fig. 2: Used capacity in percent of different room sizes over the semesters

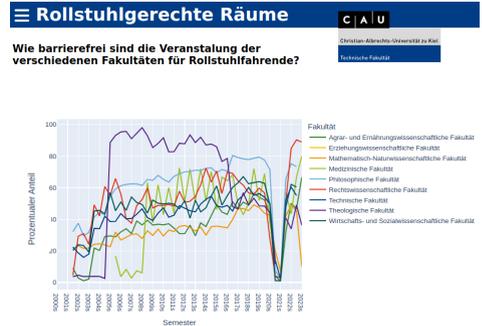


Fig. 3: Percentage of accessible rooms per department per semester

**Room sizes** The question of room planning is of ongoing interest due to the exceeding heating and lighting resources required for larger rooms, [Ki22a]. Are the rooms used to full capacity, or could events be held in smaller rooms? An excerpt from the dashboard for this topic can be seen in Figure 2. The rooms are grouped into three different categories, according to the number of people they can host. This distinction showcases that, based on the room size, there is a marked difference with respect to the used capacity. Rooms with fewer than 50 seats are often used up to 100% capacity or even more. Larger rooms (>=100 seats) on the other hand are often only 50% occupied. For this evaluation, the *turn out* parameter, a value describing the planned number of attendees for the module, is used. It is relevant to keep in mind that student attendance for lectures is often hard to estimate beforehand. Therefore, it is necessary to allow for a buffer. Based on the obtained insights, the administrative instances of the university can make an effort to determine the actual capacity of the rooms during the semester in order to determine whether this discrepancy could be a starting point for energy-saving improvements.

**Accessibility** The university strives for inclusiveness and fairness to ensure equal opportunities for all its student. This also includes wheelchair accessibility. The wheelchair accessibility of modules is documented as a flag in the database. The accessibility of the different faculties is depicted in Figure 3. The plot reveals how many faculties courses are, on average, located in wheelchair accessible rooms or even buildings. Drastic shifts in the percentage mostly correlate with changes in the locations of faculties. As an example, the Faculty of Law ('Rechtswissenschaftliche Fakultät') moved into a new building in 2021. The difference is obvious when comparing the values before and after the COVID-19 pandemic. Large jumps in the early years, e.g., in the winter semester 2004/2005 (2004w) for the Faculty of Theology ('Theologische Fakultät') suggest that the flag was not used in the years before 2005. As all events during the pandemic took place online, classified as

non-accessible, an overall drop is observable in the winter semester 2020/2021 (2020w) and summer semester 2021 (2021s).

**Walking distances** When creating lecture plans, the distances between module locations need to be taken into consideration. For this dashboard, the compulsory modules for selected semesters of two subjects of study were gathered, and the locations were visualized on an interactive map. The goal was to detect cohorts that were spread out far and evaluate whether the distances could lead to time conflicts in the semester schedules.

In Figure 4 the comparison of two chosen degree programs and semesters is depicted. Additional data on the compulsory modules for the semesters had to be acquired manually. Because this information changes frequently, it is not stored in UnivIS but recorded in the examination regulations. Manual evaluations revealed that some modules could not be chosen in certain semesters due to time conflicts. With further development, this could be expanded into a tool for automatized checking of time conflicts and added into the planning process.

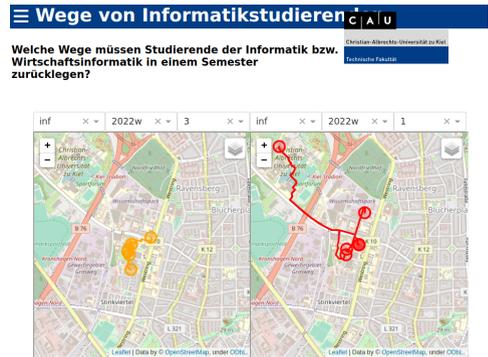


Fig. 4: Locations of compulsory modules for computer science semesters 1 and 3 in winter semester 2021/2022

**Best Prof** At Kiel University a 'Best Prof' award<sup>4</sup> is given by vote of the students in the Department of Mathematics and the Department of Computer Science every year. The dashboard traces the lectures of the past top three winners of each year. Two excerpts of the evaluated statistics are shown in Figures 5 and 6. According to these analyses, past winning professors held noticeably fewer modules on Monday, Friday, and Saturday than the other days and started their courses mostly at 8 a.m., 10 a.m., or 12 a.m.

These findings were used to predict the top three candidates of 2022. With the above-described filters, two of the three professors on the podium could be predicted correctly. The professor in third place did not fit these insights as multiple modules were held on Monday. This got the better of our logic and proved: Preferences cannot be predicted purely on statistics. If you teach well, students will appreciate it, despite the course being held on Mondays.

<sup>4</sup> <https://www.fs-infmath.uni-kiel.de/wiki/Best-Prof-Hall-of-Fame>

Beliebteste Vorlesungstage Gesamtübersicht

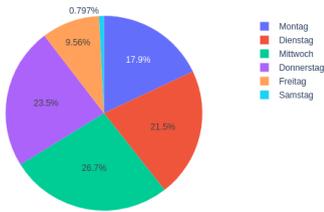


Fig. 5: Lecture days of modules from past winners

Beliebteste Vorlesungsstartzeiten Gesamtübersicht

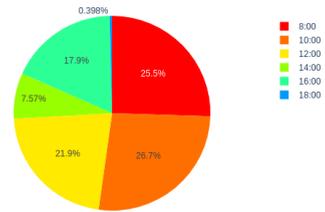


Fig. 6: Starting times of modules from past winners

## 6 Conclusion

The proposed dashboard provides insight into the evolution of Kiel University and shows the benefits from reaping the database systems in place. By using an already existing database, this project was implemented with little necessary effort on the part of the university. Thereby, this prototype is a feasible solution for interactively analyzing a large institution where the gathering of data is often complicated and faces bureaucratic difficulties.

To transfer the dashboard to other universities, it is essential to extract an equivalent data set from the system in place. Additional topics, that can be analyzed and added to the dashboard are only limited by the data available. The proposed dashboard contains a baseline of evaluations for answering the chosen questions of interest for large institutions regarding the progression of room utilization, diversity, and inclusiveness. Further evaluations are implemented in the prototype dashboard that could not be explained in detail here due to brevity. The evaluations and insights are described in German on the demo web application.

Caution has to be exercised when drawing inferences from the data because the data used are planning data and therefore do not always depict a true representation of the real conditions at the institution. A strong example is the decision to flag online events as non-accessible, resulting in a seemingly very retrogressive development concerning accessibility in the semesters during the pandemic.

As for the evaluation of Kiel University, in the demonstration, we showcased the strong suits such as the rise of wheelchair-accessible rooms, and make the potential for reasonable areas of improvement visible. Answering all chosen questions with data visualizations. We aim to make this dashboard easily transferable to other universities in the future to gather insights into their performance. Furthermore, we plan on providing additional features, such as advanced room planning, to reduce unnecessary heating costs. We are convinced that interactive platforms, like the one proposed in this work, bear the potential to pave the path for accelerated and improved performance assurance in universities.

## References

- [BL08] Baker, David P; Lenhardt, Gero: The institutional crisis of the German research university. *Higher Education Policy*, 21(1):49–64, 2008.
- [Fi20] Fischer, Christian; Pardos, Zachary A.; Baker, Ryan Shaun; Williams, Joseph Jay; Smyth, Padhraic; Yu, Renzhe; Slater, Stefan; Baker, Rachel; Warschauer, Mark: Mining Big Data in Education: Affordances and Challenges. *Review of Research in Education*, 44(1):130–160, 2020.
- [Gu20] Gutiérrez, Francisco; Seipp, Karsten; Ochoa, Xavier; Chiluíza, Katherine; De Laet, Tinne; Verbert, Katrien: LADA: A learning analytics dashboard for academic advising. *Computers in Human Behavior*, 107:105826, 2020.
- [Ki22a] Kiel University: , Energiesparmaßnahmen an der CAU. Webpage, 2022. <https://www.uni-kiel.de/de/energiesparen>, [last access 01.11.2022].
- [Ki22b] Kiel University: , UnivIS Kiel. Webpage, 2022. <https://www.univis.uni-kiel.de>, [last access 25.10.2022].
- [MJ12] Mandinach, Ellen B; Jackson, Sharnell S: Transforming teaching and learning through data-driven decision making. Corwin Press, 2012.
- [PI15] Plotly Technologies Inc.: , Collaborative data science. Webpage, 2015. <https://plot.ly>. [last access 25.10.2022].
- [STU99] Scheler, Fabian; Turowski, Stefan; Ulbrich, Peter: , Information system for universities: UnivIS. Webpage, 1999. Config Informationstechnik eG, <https://www.config.de/UnivIS/>, [last access 30.10.2022].
- [Th22] The University of Texas at Arlington (UTA): , University Analytics. Webpage, 2022. <https://www.uta.edu/administration/analytics>, [last access 01.11.2022].
- [We12] West, Darrell M: Big data for education: Data mining, data analytics, and web dashboards. *Governance studies at Brookings*, 4(1):1–10, 2012.



# Better Safe than Sorry: Visualizing, Predicting, and Successfully Guiding Courses of Study

Alexander Kerth<sup>1</sup>, Felix Schuhknecht<sup>2</sup>, Lukas Pense<sup>3</sup>, Justus Henneberg<sup>4</sup>

## Abstract:

Successfully going through a course of study is a lengthy and challenging task. To obtain a degree, many obstacles must be overcome and the right decisions must be made at the right point in time, often overwhelming students. To reduce the amount of dropouts, the goal of study advisors is to reach out to endangered students in time and to provide them help and guidance. To support the work of study advisors, who typically have to monitor a large amount of students simultaneously, we present in this demonstration an easy-to-use graphical tool that (a) allows the advisor to visualize all relevant information of study data in a responsive graph in order to overview the current study situation. In addition to visualization, our tool provides (b) a forecasting functionality based on pre-trained models and (c) a warning feature to identify endangered students early on. In the on-site demonstration, the audience will be able to step into the role of a study advisor and use our tool and all of its features to identify and guide struggling students within anonymized real-world study data.

**Keywords:** Study monitoring; Study Prediction; Visualization; Machine Learning; Graph Databases

## 1 Introduction

Successfully studying and obtaining a degree is challenging. While some students are able to successfully make it through the forest of lectures, seminars, and labs on their own, many struggle in finding the right individual path by themselves. To counter this problem, many universities employ so-called *study advisors*. Their task is essentially threefold: (1) To identify struggling students, for instance by analyzing their performance over the past semesters. (2) To provide guidance for the identified students on how to improve their individual situation. (3) To monitor whether the guidance actually helps and improves the situation of the students. With their work, study advisors help in reducing (avoidable) dropouts and eliminate stress and uncertainty on the side of students.

Unfortunately, all three tasks are highly difficult for the study advisor in the present situation. In terms of (1), typically, a large number of students must be monitored, namely all currently enrolled students in all stages of their studies. Therefore, overseeing the sheer amount

---

<sup>1</sup> Johannes Gutenberg University Mainz, Institute of Computer Science, Staudingerweg 9, 55128 Mainz, Germany  
alkerth@students.uni-mainz.de

<sup>2</sup> schuhknecht@uni-mainz.de

<sup>3</sup> pense@uni-mainz.de

<sup>4</sup> henneberg@uni-mainz.de

of data is already challenging when done (pseudo-)manually. Connected to this is the problem that the study data is typically materialized in some sort of course management system (CMS). The primary purpose of such a database is to reliably log all entered study data, but not to perform any form of sophisticated analysis on it. This leads to the unpleasant situation that accessing and analyzing the existing study data is already a cumbersome step for the study advisor. Assuming that struggling students have been identified somehow in step (1), step (2) is even more challenging. Now, the study advisor has to come up with guidances and recommendations that fit to the individual problems of each student. This requires a deep individual analysis of the study performance so far and the design of a fitting counter-steering measure. Here, different courses could be advised to be taken in a specific order. The optimization goal is basically to find a suitable trade-off between progress, interests, and pressure for the student. When guidance has been given to a struggling student, in step (3), the performance of the student must be monitored and evaluated throughout the next semesters in some sort of feedback loop. Here, the study advisor has to carefully analyze the development of the student over a longer period of time, requiring cumbersome (repetitive) data analysis and comparison.

To support study advisors in their challenging tasks, in the following we propose a tool to assist their workflow in all three previously mentioned steps. Note that we specifically use the term “*assist*” and not “*replace*”, as we believe that human advising is (and will always be) essential in this sensitive field. Precisely, our tool assists the advisor in the following aspects:

(a) It makes study data accessible for the study advisor by connecting students, lecturers, courses, and exam results with each other and by presenting their relation in a *graph-based visualization*. We provide different views for different analysis purposes: For example, it is possible to visualize for a particular student all taken courses and exam results. Or it is possible to focus on a specific course and to see all exam performances of a specific semester or over multiple semesters for all students. Any available meta-data, such as passing/failure ratio, is also presented to ease the interpretation of the data.

(b) It contains an *early warning system* that automatically identifies potentially struggling students. To do so, it predicts the likelihood of finishing the studies using an ML model, which has been trained on available study data, and combines it with the number of exams failed so far. The generated list of students can be ordered and post-filtered by the study advisor to contact the students in danger.

(c) It allows the advisors to create a *course of study forecast* for a particular student to give recommendations on which courses to take. This can come in handy if the curriculum contains many courses to choose from. Again, we use pre-trained ML models to generate this forecast and visualize it in an easy-to-digest manner. Besides the proposed courses, the forecast also contains the predicted grade range and passing probability for each course.

Note that we designed our tool to be usable by *non-computer-scientists*, i.e., there is no

programming or query writing required. Therefore, our tool can be applied university-wide by study advisors of all faculties.

## 2 Architecture and Setup

Our tool runs in a web browser and uses a multitude of technologies: For the frontend, we use a combination of PHP, JavaScript, and Bootstrap, running on an Apache web server. The backend is twofold: To manage the data, we use Neo4j [ht22] as a graph-based database management system that is queried by the frontend via Cypher [Fr18] queries. Since all study data in the CMS is typically in relational format, we use a converter that translates the relational data into the corresponding graph representation by turning foreign-key relationships into edges. In our case, we extracted the (anonymized) study data in relational format from JoGu-StlNe [Jo22], the CMS of JGU Mainz, and then converted the data into a corresponding graph representation.

The machine learning part, which is required for forecasting and warning, is realized in Tensorflow and largely builds upon the N-RELAGGS [PK19] work. The core idea is to feed a neural network with input features composed of joined study data. The joined data contains information about enrollment, semester statistics, and exam results for each student. Underneath, Tensorflow and Keras are used to train the model and to forecast the performance of students of interest. Note that other works [Zh20, Má13] also tried to predict the performance of students using data mining and ML-assisted approaches. In general, these approaches could be integrated into our prediction backend as well.

## 3 Visualization

Figure 1 shows a screenshot of the main view of our tool running in a browser. The depicted center view shows the visualization of the performance of a specific student (turquoise node). Around the student, all taken courses of that student are arranged (blue nodes) with the semester in which the course was taken (yellow nodes). Linked to each course/semester combination is the grade of the corresponding exam (green nodes for passing, red node for failing). The border of each course node additionally shows the passing/failing ratio of this particular course over all semesters. The same applies to exam nodes.

Note that at any time, the user can click a node in the center view to focus the view on that node. Focusing on a node does more than rearranging the currently visible nodes as it might trigger the loading of new nodes. For example, when clicking on a specific exam, the center view will load and show all students of that particular exam. To go back to the previous view, the user can click on the “return” button on the top of the view at any time.

Alongside the center view, on the right side we list various statistics that support the current visualization. For example, we show the average grade of the student, the fraction of passed exams, the number of courses taken, and the passing ratio of all exams. Note that the

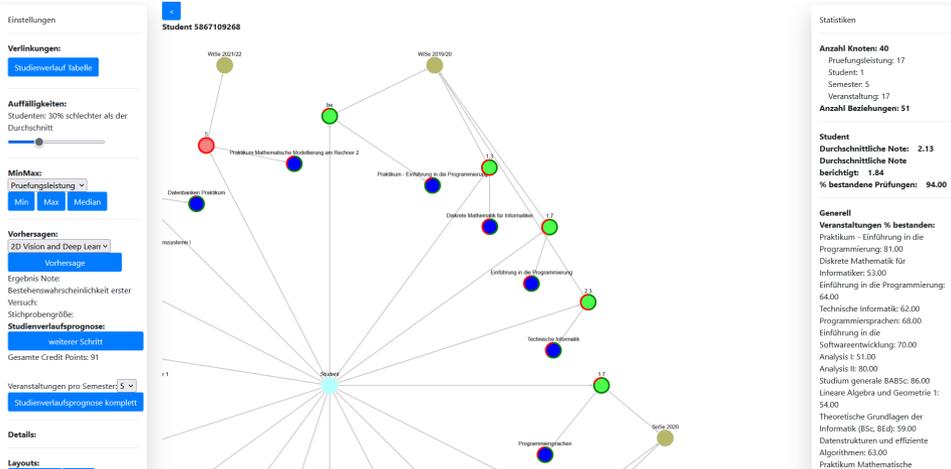


Fig. 1: Frontend overview of our tool running in a web browser.

right-side view automatically adapts to the current visualization of the center view. The left view of the tool serves as a settings and action panel that can be used to adjust the current visualization and to trigger the loading of new visualizations. This includes triggering the the course of study forecast (Section 4) and the early warning system (Section 5).

## 4 Course of Study Forecast

Besides visualization, our tool supports forecasting the performance of a particular student in a particular course or even the whole (remaining) course of study. To enable this feature, the course of study regulations must be encoded in our tool in advance, which contain information on which courses are available and which combination of course (types) must be passed to obtain the degree. For this demo, we encoded the study regulations of the B.Sc. in Computer Science (2016 version) of JGU Mainz.

We now trained four different models for each course on the available study data, where each model is tailored towards a different stage of study. We train the first model on the first semester study data of all students, whereas the second model is trained on the first *and* second semester study data of all students, and so on. We consider only courses that contained at least 40 students and feed the model with exam results as well as meta-data such as achieved credit points per semester and achieved credit points overall. When forecasting for a particular student, the system automatically picks the model that fits best to the semester in which the student is currently enrolled. For each recommended course, the model predicts one of the outcomes “good” (grade range 1-2), “satisfactory” (grade range 3-4), or “failed” (grade 5).

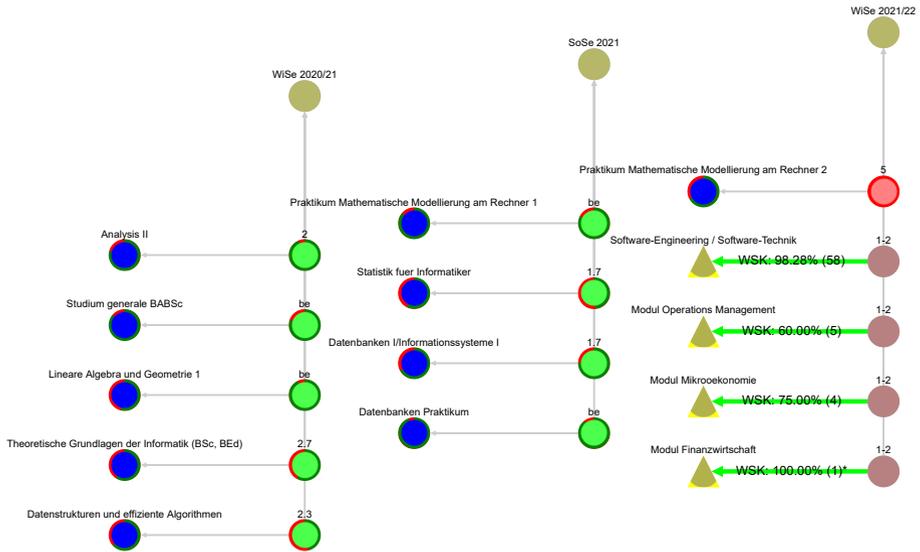


Fig. 2: Course of study forecast for a particular student for winter semester 2021/2022 based on the performance of the previous two semesters.

Additionally, a passing probability is predicted, which might be more relevant for endangered students than the actual grade. It is based on a Naïve Bayes classifier, which has been used in this context successfully in [DG17, PP21], using one of two metrics: In the first variant, which is used by default, it computes the probability of passing a particular exam in the first try depending on how many previous exams were passed in the first try. In the second variant, which is used if the sample size is too small, it computes the passing probability depending on the results achieved in already taken exams. Figure 2 shows how the complete forecast visualization looks like. For the student of interest, we show for each semester the already taken exams (winter semester 20/21 and summer semester 21 in this example) as well as the predicted semester (winter semester 21/22 in this example). For the predicted semester, we show all recommended courses along with the predicted grade ranges, passing probabilities, and sample sizes.

## 5 Early Warning System

In addition to forecasting the performance of a particular student, our tool also supports the automatic identification of potentially struggling students, so that the study advisor can help these students as early as possible.

To do so, we again utilize our four pre-trained models and predict for each currently enrolled student whether he or she will acquire the degree or not. We then combine this prediction with the amount of already failed exams to compute a “risk score”, by which we can order all students. The user can then view all students whose risk score lies above a certain threshold in the frontend, as shown in Figure 3. From this visualization, the study advisor can then directly perform a closer inspection of the performance of the endangered students.

| Gefährdung | Student    | Ø Note | Fachsemester | Studiengang         | Link                           |
|------------|------------|--------|--------------|---------------------|--------------------------------|
| 9.83       | 4579990613 | 4.17   | 8            | Bachelor Informatik | <a href="#">Visualisierung</a> |
| 9.71       | 4874479461 | 5.00   | 4            | Bachelor Informatik | <a href="#">Visualisierung</a> |
| 9.70       | 5237004224 | 4.23   | 6            | Bachelor Informatik | <a href="#">Visualisierung</a> |

Fig. 3: List of “endangered students”, who have a high risk score.

Note that we also support the visualization of “risky courses” that seem to be related to dropouts. Here, we define risk as the number of students who failed an exam and then dropped out in relation to the total number of students who took the corresponding course. Figure 4 shows an example list representation of such identified courses.

| Veranstaltung                                  | Anzahl gescheiterter Studenten | Anzahl Studenten an Veranstaltung teilgenommen | Anteil |
|--|--------------------------------|--|--------|
| Einführung in die Höhere Mathematik (Analysis) | 18                             | 90   | 20.00% |
| Elementare Algebra und Zahlentheorie           | 28                             | 204  | 13.73% |
| Diskrete Mathematik für Informatiker           | 93                             | 701  | 13.27% |

Fig. 4: List of “risky courses” that are potentially related to dropouts of students.

## 6 Demonstration

In our on-site demonstration, the users will be able to freely interact with our tool, which we pre-load with anonymized real-world student data from the computer science B.Sc. course of study of JGU Mainz. The audience will step into the role of a study advisor and experience the entire tool-assisted workflow: Identifying potentially endangered students via the warning system, exploring their individual problems by navigating through the different visualizations, and finally predicting the best courses to take alongside with their chances of success. By inspecting the courses year-by-year, the audience can also identify temporal trends in the passing/failing ratio and the number of students visiting the course.

**Acknowledgements:** We would like to thank Hans-Jürgen Schröder for his contributions to early stages of this work and his efforts in acquiring the anonymized study data. Also, we would like to thank JGU Mainz for providing the anonymized study data, as well as Markus Blumenstock for his input as a study advisor.

## Bibliography

- [DG17] Dake, Delali Kwasi; Gyimah, Esther: Students Grades Predictor using Naïve Bayes Classifier – A Case Study of University of Education, Winneba. *International Journal of Innovative Research in Science, Engineering and Technology*, 6(10), 2017.
- [Fr18] Francis, Nadime; Green, Alastair; Guagliardo, Paolo; Libkin, Leonid; Lindaaker, Tobias; Marsault, Victor; Plantikow, Stefan; Rydberg, Mats; Selmer, Petra; Taylor, Andrés: Cypher: An Evolving Query Language for Property Graphs. In (Das, Gautam; Jermaine, Christopher M.; Bernstein, Philip A., eds): *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, pp. 1433–1445, 2018.
- [ht22] <https://neo4j.com>, 2022.
- [Jo22] Johannes Gutenberg University Mainz: , <https://jogustine.uni-mainz.de/>, 2022.
- [Má13] Márquez-Vera, Carlos; Cano, Alberto; Romero, Cristóbal; Ventura, Sebastián: Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data. *Appl. Intell.*, 38(3):315–330, 2013.
- [PK19] Pensel, Lukas; Kramer, Stefan: Forecast of Study Success in the STEM Disciplines Based Solely on Academic Records. In (Cellier, Peggy; Driessens, Kurt, eds): *Machine Learning and Knowledge Discovery in Databases - International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*. volume 1167 of *Communications in Computer and Information Science*. Springer, pp. 647–657, 2019.
- [PP21] Perez, Joann Galopo; Perez, Eugene S.: Predicting Student Program Completion Using Naïve Bayes Classification Algorithm. *International Journal of Modern Education and Computer Science (IJMECS)*, 13(3):57–67, 2021.
- [Zh20] Zhao, Yijun; Xu, Qiangwen; Chen, Ming; Weiss, Gary: Predicting Student Performance in a Master’s Program in Data Science using Admissions Data. In (Rafferty, Anna N.; Whitehill, Jacob; Romero, Cristóbal; Cavalli-Sforza, Violetta, eds): *Proceedings of the 13th International Conference on Educational Data Mining, EDM 2020, Fully virtual conference, July 10-13, 2020*. International Educational Data Mining Society, 2020.



# JPTest - Grading Data Science Exercises in Jupyter Made Short, Fast and Scalable

Eric Tröbs,<sup>1</sup> Stefan Hagedorn,<sup>2</sup> Kai-Uwe Sattler<sup>3</sup>

**Abstract:** Jupyter Notebook is not only a popular tool for publishing data science results, but can also be used for the interactive explanation of teaching content as well as the supervised work on exercises. In order to give students feedback on their solutions, it is necessary to check and evaluate the submitted work. To exploit the possibilities of remote learning as well as to reduce the work needed to evaluate submissions, we present a flexible and efficient framework. It enables automated checking of notebooks for completeness and syntactic correctness as well as fine-grained evaluation of submitted tasks. The framework comes with a high level of parallelization, isolation and a short and efficient API.

**Keywords:** Jupyter; Teaching; Exercising; Unit-Testing; Automation

## 1 Motivation

Teaching programming languages, SQL or data science related concepts often involves exercises in which students have to solve tasks by writing programs and queries on their own. Often, these exercises need to be evaluated and graded by some faculty member. However, with hundreds of students, the grading process quickly becomes a burden and often leads to the fact that only a sample is checked or that the number of tasks for the students is reduced. However, especially the latter is to the disadvantage for the students as they miss the potential of exhaustive examples for practicing with valuable feedback.

Thus, in order to exploit the possibilities of online and remote learning as well as to reduce the burden of manually coding related tasks, our goal is to provide an extensive framework to distribute and evaluate programming tasks. Especially for data analytic tasks Jupyter Notebooks<sup>4</sup> have become very popular as they allow to mix formatted text with executable code. Notebooks are also useful for teaching purposes as they allow to show descriptions and tasks within the same file. In our case, the notebooks are used in the context of a data science lecture which contains exercises after every chapter to repeat what has been learned.

In order to give students feedback on their solutions, it is necessary to check and evaluate the submitted work. In our case, this is compounded by the fact that not all assignments

---

<sup>1</sup> Technische Universität Ilmenau, Germany, Eric.Troeb@tu-ilmenau.de

<sup>2</sup> Technische Universität Ilmenau, Germany, Stefan.Hagedorn@tu-ilmenau.de

<sup>3</sup> Technische Universität Ilmenau, Germany, kus@tu-ilmenau.de

<sup>4</sup> <https://jupyter.org/>

are submitted at the same time and thus there is no practice effect on those who evaluate solutions. At the same time, multiple attempts might be allowed and we use tasks that are complicated to grade manually, for example, when a lot of `if`-statements need to be used.

In this paper, we present a flexible and efficient framework, called *JPTest*<sup>5</sup>, to automatically evaluate and grade code from Jupyter notebooks. The main goal in developing *JPTest* was therefore to automate the evaluation of coding tasks, while creating a tool that can also check notebooks for completeness and syntax errors. The focus during development was on fast execution through parallelization, isolated execution of student code and an efficient interface. Most of our tasks can be evaluated by classical unit testing of single functions or by comparing manipulated data sets with those of a sample solution. To shorten the process with these task types, annotations exist to express recurring parts of the tests.

## 2 Related Work

Although the lockdown of schools and universities has drastically increased the need for online learning formats, especially in computer science related lectures, various automated solutions have been created and used for years. However, these are often self-implemented solutions, that are not available to other groups or lack features important for grading. During the peak of the lockdown-induced remote learning phase, the database community presented some of their solutions and experiences with remote learning approaches in the *Datenbank Spektrum* journal.

First among these is *SQLValidator* by Obionwu et al., where students can easily submit queries to a prepared database and receive detailed feedback and explanations of mistakes they encountered. *SQLValidator* also includes the possibility to create questionnaires and test students automatically. Even though the authors report positive effects on their courses, the software is limited to SQL [Obi+21].

The second example we would like to mention is a Data Engineering course for 10,000 participants by Alder et al. Based on the openHPI platform of the Hasso Plattner Institute in Potsdam, a so-called *Massive Open Online Course* is offered, which includes lectures supplemented by videos, homework and exams. According to the number of participants, evaluation by hand is nearly impossible. Automated correction was made possible by the use of multiple-choice and multiple-answer questions [Ald+20].

Beyond these teaching related approaches and because Jupyter is widely used not only for prototyping and ad-hoc analytics, there also exist test frameworks for notebooks.

*papermill*<sup>6</sup> is a project to parameterize notebooks. It works by evaluating tags and allows modifying, storing, inspecting and running notebooks with different sets of parameters.

---

<sup>5</sup> <https://github.com/erictroebs/jptest>

<sup>6</sup> <https://github.com/nteract/papermill>

Although not directly related to our work, it might be a valuable alternative to create similar reports with different values from a single notebook file.

*nbgrader*<sup>7</sup> is an integrated solution for grading Jupyter Notebooks. It can be fully operated via a graphical interface and also allows mixing manually and automatically graded tasks. It also allows generating student versions of an assignment. In contrast to *nbgrader*, JPTest completely detaches tests from notebooks, allows runs to test syntax and completeness, and does not require any plugins in Jupyter. In addition, JPTest allows more freedom in the design of the test code, for example through setup and teardown methods.

### 3 System Description

JPTest is an unit testing framework for Jupyter Notebooks created with the needs of our data science lecture in mind. It uses *nbclient*<sup>8</sup> as a base for executing code in notebooks. *nbclient* was originally created for running notebooks to get the output of the cells and process it, for example, for conversion to other formats. As in Jupyter Notebook, a kernel is necessary for the execution of each code cell. It is not strictly necessary to start a separate kernel for each notebook. However, JPTest does just that. From the process that was started to execute the tests, at least a single kernel is started for each test. Each kernel is executed in a separate process so that proper multiprocessing is possible across all tasks, while the test process acts as a coordinator. JPTest always runs on an in-memory copy of the notebook and does not modify files, but tests and code in the notebook still have the possibility to do so.

In summary, there is a process in which the test code runs and from which kernels are started. We refer to this coordinator as the test context. Since our tests contain parts of the solution, it is important that they are managed and stored separately from the notebooks. Each test has exactly one function, which is identified by an annotation. Multiple tests can be collected in one or more files and run together, adding up the scores and collecting the comments. On the other hand, a separate Python process exists for each kernel, which we refer to both individually and as a set of all these processes as the notebook context. Figure 1 visualizes the relationship and communication structure between the created processes.

For communication between contexts the default implementation *jupyter\_client* is used, resulting in the use of ZeroMQ. Pickle is used to serialize the objects to be transferred, which, in contrast to JSON for example, can also serialize more complex objects such as NumPy Arrays or Pandas DataFrames. Code written for the test context relies heavily on the asyncio framework to exploit this multiprocessing environment.

The easiest way to execute code in the notebook is via the *cells* property. It returns a list of all cells present in the notebook and allows to filter and execute them one by one. The function

<sup>7</sup> <https://github.com/jupyter/nbgrader>

<sup>8</sup> <https://github.com/jupyter/nbclient>

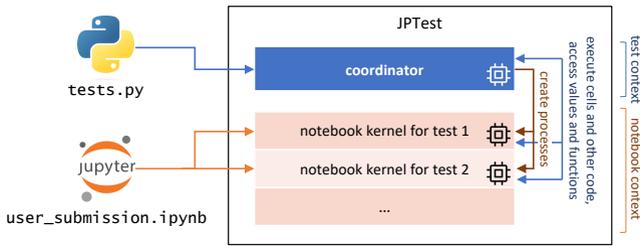


Fig. 1: The coordinator loads the tests and creates at least one kernel in a separate process for each test. The kernel processes communicate only with the coordinator.

`execute_cells` represents a shortcut to select only code cells by tags prior to executing them in their order of appearance.

Besides executing single cells, it is also possible to interact with objects and code in the notebook context. The most important class in this regard is *NotebookReference*. References returned, for example, by the `ref` and `get` functions, represent objects in the notebook context and may be used for interaction in various ways. For example, they can be serialized and transferred to the test context. However, it is also possible to create sub-references to attributes or keys. References to functions can also be called, where the parameters can be either other references or local variables. In the latter case, these are serialized and sent to the notebook kernel before being called.

The result of almost all operations is delayed. This prevents the need for nesting of `await` statements, what really enhances the readability, and improves the performance by less inter-process communication. Only with a call to `receive` or `execute` the final statement is built and executed in the notebook context, which can cause errors to occur later than their actual call.

Furthermore, functions in the notebook context may be replaced with others, for example to skip network requests and return a fixed response instead to speed them up. Even more interesting is the monitoring of functions, where calls with their parameters and return values are tracked. This makes it possible, for example, to determine whether a user's implementation is using recursion. The available context managers provide the option of replacing functions only for specific statements.

It is also possible to inject own code from the test context into the notebook context. The most simple way is to use a string that is inserted as a new cell at the end of the notebook and executed in the notebook context. However, there are also two methods to inject functions: The first transfers a function to the notebook context and returns a reference. This can be called as described before or passed as a parameter to another function. The second copies the body of a function into a new code cell and executes it. The header is used exclusively in the test context. We use this functionality to write syntactically correct code with all the

user\_submission.ipynb

```
[ ]: # task-1
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

tests.py

```
1 from jptest2 import *
2
3 @JPTest('Task 1', max_score=1)
4 async def test_task1(nb: Notebook):
5     await nb.execute_cells('task-1')
6
7     fib_fun = nb.ref('fib')
8     yield (
9         await fib_fun(5).receive() == 5,
10        1,
11        'better luck next time',
12        'very good'
13    )
```

Fig. 2: Left: A student's submitted fibonacci function. Right: A unit test that executes all cells with the tag `task-1`, creates a reference to the `fib` function and awards one point if a call to this function with the parameter 5 returns 5.

benefits of analysis within an IDE, although it is later only executed in the notebook context, which is a massive advantage over writing code as a string. The parameters in the header define the necessary variables that are present in the notebook through the execution of previous cells.

To register tests different annotations are used. They are available to either prepare one or two notebooks or to run specific cells and copy single variables into the test context. A maximum number of points can be set as well as an execution timeout. This reduces the writing of tests to as less code as possible. The assignment of points is done using the `yield` keyword. The test function then works as a generator, where each returned value is understood as a part of the score. The value consists of a tuple containing a condition, a score to be awarded when it is met and optional comments on success or failure. Figure 2 shows a basic unit test which uses this concept.

Last but not least, it is possible to connect other kernels, but Python-specific features are lost in this process. Currently SQLite and DuckDB are partly supported.

## 4 Best Practices

Regarding references, there is also a way to exchange values between notebooks. The function `store` can be used to store values, but also references into notebooks. References can also be used as parameters to call functions within the notebook. If the reference is from the notebook where it should be used, this operation is trivial. If it is from another notebook, it automatically is copied to the former. However, copying objects across notebooks should be avoided where possible.

In general, the test context can become a bottleneck because it uses only one thread. To use the performance of multiple cores, the notebooks should work as independently as possible and the test context should only be used for coordination and evaluation. For example, when we evaluate manipulated DataFrames, one notebook runs the student's solution and one runs

the sample solution. Both resulting DataFrames are copied to the test context and checked there only for equality, so that the computationally intensive operations are outsourced.

In the case of data loading, we use setup functions to modify the data set before starting the tests. Depending on the task, only a portion of all data is selected or the data set is modified. This ensures that the student's code actually solves the problem in general, rather than exclusively providing the answer for the given data. A reduced data set can also speed up the execution.

The simplest test possible is the one where no test file is provided. In this case JPTTest loads a default implementation that executes all cells once in the correct order, does not score and passes exceptions. This can be used to check notebooks for syntax errors, determine if libraries are missing within an image or if data sets have not been shipped.

Usually we use Docker to create a reproducible environment for our tests. All necessary dependencies are installed in the image, while required data sets are mounted read-only. By testing a notebook within the container after it has been modified, we can determine whether the image does actually include all the required dependencies. In addition, the containers operate without an internet connection, which creates an isolated environment for each user's notebook.

## 5 Demo Contents

We provide several Jupyter Notebooks and data sets to create and change unit tests using a set of tasks from our data science lecture, which mainly relies on Python. This includes submissions where functions are tested by simple unit tests as well as comparisons of objects with the results from sample solutions. In addition, we show how data sets can be modified before testing to reject hard-coded solutions and how function replacements can speed up execution times. We also handle cases where values found experimentally by our students have to be taken into account.

At the same time, using `asyncio`, the communication between tests and the concurrently running notebooks will be explained further. Furthermore, we include test that make use of the API to grade common tasks in just a few lines of code. As there is no way to avoid explaining and using database systems in our lectures, we take a look at the use of external services and how we try to replace them using embedded software.

Last but not least we show the use of the test framework with other kernels, so that SQL statements, for example, can be graded directly.

**Acknowledgements.** This work was partially funded by the German Federal Ministry of Education and Research under grant no. 16DHBKI085.

## References

- [Ald+20] Nicolas Alder et al. “Ein Data Engineering Kurs für 10.000 Teilnehmer”. In: *Datenbank-Spektrum* 21.1 (2020), pp. 5–9.
- [Obi+21] Victor Obionwu et al. “SQLValidator - An Online Student Playground to Learn SQL”. In: *Datenbank-Spektrum* 21.2 (2021), pp. 73–81.



# MEDUSE: Interactive and Visual Exploration of Ionospheric Data

Joshua Reibert,<sup>1</sup> Arne Osterthun,<sup>2</sup> Marcus Paradies<sup>3</sup>

**Abstract:** Spatio-temporal models of ionospheric data are important for atmospheric research and the evaluation of their impact on satellite communications. However, researchers lack tools to visually and interactively analyze these rapidly growing multi-dimensional datasets that cannot be entirely loaded into main memory. Existing tools for large-scale multi-dimensional scientific data visualization and exploration rely on slow, file-based data management support and simplistic client-server interaction that fetches all data to the client side for rendering.

In this paper we present our data management and interactive data exploration and visualization system MEDUSE. We demonstrate the initial implementation of the interactive data exploration and visualization component that enables domain scientists to visualize and interactively explore multi-dimensional ionospheric data. Use-case-specific visualizations additionally allow the analysis of such data along satellite trajectories to accommodate domain-specific analyses of the impact on data collected by satellites such as for global navigation satellite systems and earth observation.

**Keywords:** Exploratory Data Analysis; Ionospheric Data; Data Cubes

## 1 Introduction

The Earth's ionosphere is observed by an ever-growing amount of sensors—both ground- and space-based—which results in a giant corpus of raw data [Ca20]. The research of this space weather is important since space weather events (e.g., solar storms) can affect the quality of satellite communication which in turn can impact services like global navigation satellite systems (GNSS) such as the widely used global positioning system (GPS) [SJH19]. Hence, scientists have taken on harmonizing and integrating the raw measurement data into dense models of the ionosphere to relate noise in satellite-based data back to space weather.

The resulting spatio-temporal model data quickly becomes unwieldy as the data volume grows, especially in the temporal dimension. Hence, researchers oftentimes work on large sets of small files and resort to static visualizations of small areas of interest which is cumbersome and limits interactivity during data exploration. Interactive visualization allows

---

<sup>1</sup> German Aerospace Center (DLR), Institute of Data Science, Mälzerstr. 3 – 5, 07745 Jena, Germany joshua.reibert@dlr.de

<sup>2</sup> German Aerospace Center (DLR), Institute of Data Science, Mälzerstr. 3 – 5, 07745 Jena, Germany arne.osterthun@dlr.de

<sup>3</sup> German Aerospace Center (DLR), Institute of Data Science, Mälzerstr. 3 – 5, 07745 Jena, Germany marcus.paradies@dlr.de

users to visually explore the data and quickly gain new insights but it highly depends on the system's latency which impacts analysis performance [LH14]. As a consequence, adequate data management technologies to store and access the spatio-temporal model data are required to support scientists in their tasks. Besides providing fast access to the data in the data management system, well-known techniques to reduce latency, e.g., client-side caching, binary & compressed data transfers, progressive visualization, and execution of native code on the client side, reduce the overall execution time of interactive exploration queries [BS21]. To the best of our knowledge, currently no such data management and interactive exploration system for spatio-temporal models exists. VirES<sup>4</sup> provides some of the features but is focused on specific satellite products and is hard to extend to model data.

In this paper we present an initial design of the MEDUSE eco system consisting of a data management backend and a data exploration and visualization component specifically designed to interactively and visually explore and analyze ionospheric model data. MEDUSE builds on a custom data backend to query metadata and provide fast access to the underlying multi-dimensional data exposed through a data cube data model. The data backend decouples the physical data layout, i.e., files in specific data formats, such as NetCDF, TileDB, or Zarr, from the logical layout and data model based on the concept of *data cubes* [Ba17]. The model data is visualized along the spatial and temporal dimensions—each visualizing individual slices of the data. This allows users to iteratively navigate the data which is adaptively loaded to reduce latency and provide fluid interaction. Specialized features like showing data along satellite trajectories are tailored to expert domain users that want to analyze the ionosphere's impact on satellite communications.

## 2 Background

The ionospheric models in our use case are developed by domain experts from the atmospheric sciences [HJP22]. They are based on ground- and space-based measurements to compute a dense grid of earth-centric electron density data. In addition to the two common spatial dimensions of longitude and latitude, this also includes the vertical altitude dimension and a temporal dimension, thereby resulting in a four-dimensional dataset.

The model includes the *electron density* ( $N_e$ ) in the ionosphere with these four dimensions and three additional, derived variables which omit the altitude dimension and are hence only three-dimensional: The *vertical total electron content* (VTEC) is the integral of the electron density along the altitude dimension. The maximum electron density along the altitude dimension is the *peak density* ( $N_mF2$ ) and the corresponding altitude the *peak density height* ( $h_mF2$ ). The spatial resolutions are relatively low with 72 values for longitude and latitude and 112 altitude steps. However, the data can be computed for up to 5-minute intervals which quickly adds up when larger time spans are processed. While the data for a single point in time makes up 2.5 MiB, a year worth of data amounts to about 266 GiB. The

<sup>4</sup> <https://earth.esa.int/eogateway/tools/vires-for-aeolus>

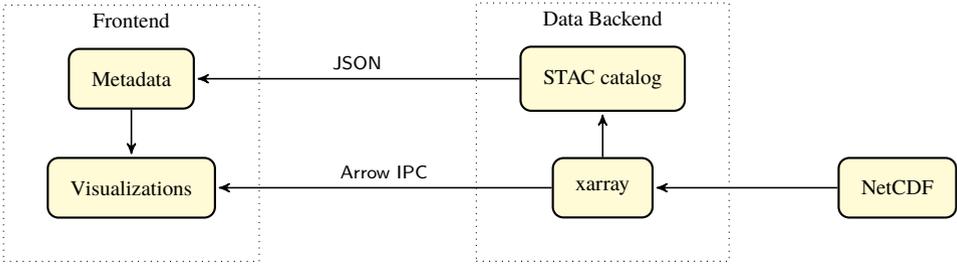


Fig. 1: MEDUSE architecture overview.

data size is then again multiplied by the number of models that are considered. Currently, we consider two models in our application.

### 3 Interactive Visual Analysis of Ionospheric Models in MEDUSE

Figure 1 depicts the overall architecture of MEDUSE consisting of a web-based client and the data management backend storing all the model data.

#### Data Backend

The data for the ionospheric models are provided in the NetCDF format from which meta-data is generated in the spatio-temporal asset catalogs (STAC)<sup>5</sup> format in a pre-processing step. It encompasses data dimensions, axis types, coordinates, extreme values, and free-form metadata, such as textual descriptions. A custom backend provides endpoints to query this STAC metadata, queries for dicing and slicing using dimension ranges as well as point queries for specific dimension value tuples. All of the queries are validated against the metadata. The actual data access is performed using xarray<sup>6</sup> and results are sent over the network in the inter-process communication (IPC) format of Apache Arrow<sup>7</sup> via HTTP. Furthermore, aggregations can be queried directly by specifying the operation to compute and the dimensions to apply them on. To support querying along satellite trajectories, point queries interpolate the model data linearly.

#### Frontend

We developed a web application that builds on the metadata and the data cubes to enable analysts and other domain users to quickly and interactively explore the ionospheric data.

<sup>5</sup> <https://stacspect.org>

<sup>6</sup> <https://docs.xarray.dev/>

<sup>7</sup> <https://arrow.apache.org/>

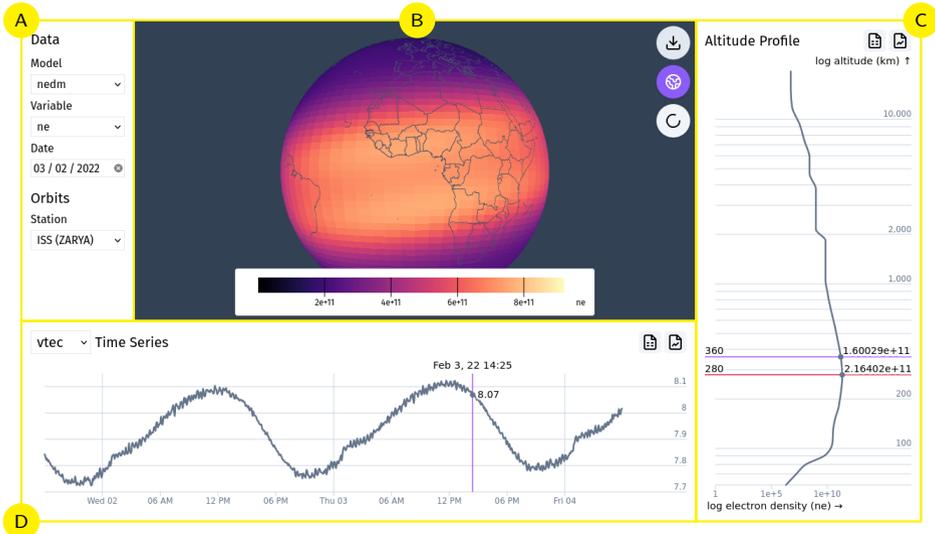


Fig. 2: The web application visualizes the ionospheric data along the different dimensions. Different models, variables, and dates in that dataset can be selected (A). The map visualizes the selected variable along the spatial dimensions (B), the altitude profile visualizes the electron density along the vertical altitude dimension (C), and the time-series chart visualizes a variable along the temporal dimension (D).

Figure 2 depicts a screenshot of the interactive data exploration and visualization tool of MEDUSE. It features a map or globe visualization for the horizontal spatial dimensions as well as an altitude profile for the vertical spatial dimensions, and a time-series chart for the temporal dimension. All of these visualizations are linked such that users can select specific dimension values in them, for example a certain timestamp in the time-series, and the other visualizations will update accordingly. As a result, users can directly and interactively select dimension values in the visualizations to explore regions of interest in the data instead of relying on external widgets.

Users can initially select one of the provided models, a variable, and a date of interest (see Figure 2 (A)). The central visualization shows the color-encoded data along the horizontal spatial dimensions mapped to Earth with country borders to give a frame of reference (see Figure 2 (B)). It can be toggled between a 3D globe view and 2D map with a dedicated button and uses WebGL for real-time rendering and interaction. Users can select individual grid cells to focus on that spatial region. The altitude profile visualizes the electron density for the selected timestamp along the altitude dimension (see Figure 2 (C)). It highlights the peak density value as well as the respective altitude and can be used to interactively inspect altitude and density values by hovering, and to select a different altitude value. The time-series visualizes how a selected variable changes over time in a 60 hours window around the selected date (see Figure 2 (D)). Again, specific values and the respective timestamp are

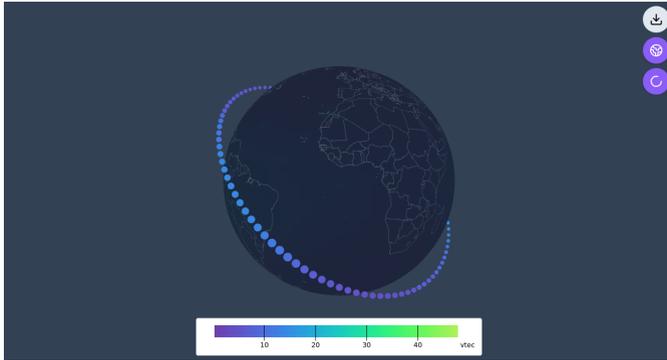


Fig. 3: The satellite trajectory visualization shows the trajectory of a selected satellite and color-codes variable values along it. The time-series chart simultaneously plots the data as a line chart.

shown when hovering the chart and also allow interactively selecting a different timestamp that is propagated to the other visualizations.

Since the electron density affects the quality of satellite communications, satellite operators and users of GNSS or earth observation are often interested in investigating electron density along satellite paths. We accommodate this use case by including a set of satellites and allowing users to select one. Satellite data is provided as two-line element set (TLE) which is used to compute the earth-centric 3D position from given timestamps. The resulting set of values for all four dimensions is used to query the data from the data cube. The satellite trajectory can then be toggled to be shown on the map or the globe and again color-coding data values along its path (see Figure 3).

The application aims to allow researchers working with ionospheric data in validating their models and investigating effects of the electron density on collected data from satellites. However, this is often just a first step to discover relations and anomalies that entail further, more focused analyses. Hence, we also provide the functionality to export the underlying data of visualizations for further inspection and the charts as images for presentation.

## 4 Demonstration Outline

For the demonstration, we will assume the role of a researcher working with earth observation data who has encountered unusual levels of noise in their data. We will use the web application to interactively explore the ionospheric models to look for unusual ionospheric activity. Finally, we will take a look at the satellite's path to analyze the electron density along its path and identify anomalies.

## 5 Summary & Outlook

In this paper we presented the initial design of MEDUSE, a data management and interactive exploration and visualization system for spatio-temporal model data from the ionosphere. In the future, we want to continue to improve our prototype to reduce latency, e. g. by using web assembly for client-side data cube access and computations as well as WebSockets to reduce network latency. Furthermore, we want to generalize the data backend to be more generally applicable to data-intensive web applications working with multi-dimensional data beyond atmospheric sciences.

## References

- [Ba17] Baumann, P.: Standardizing Big Earth Datacubes. In (Nie, J.; Obradovic, Z.; Suzumura, T.; Ghosh, R.; Nambiar, R.; Wang, C.; Zang, H.; Baeza-Yates, R.; Hu, X.; Kepner, J.; Cuzzocrea, A.; Tang, J.; Toyoda, M., eds.): 2017 IEEE International Conference on Big Data (IEEE BigData 2017), Boston, MA, USA, December 11-14, 2017. IEEE Computer Society, pp. 67–73, 2017.
- [BS21] Battle, L.; Scheidegger, C.: A Structured Review of Data Management Technology for Interactive Visualization and Analysis. *IEEE Trans. Vis. Comput. Graph.* 27/2, pp. 1128–1138, 2021.
- [Ca20] Carley, E. P.; Baldovin, C.; Benthem, P.; Bisi, M. M.; Fallows, R. A.; Gallagher, P. T.; Olberg, M.; Rothkaehl, H.; Vermeulen, R.; Vilmer, N.; Barnes, D.; the LOFAR4SW Consortium: Radio Observatories and Instrumentation Used in Space Weather Science and Operations. *J. Space Weather Space Clim.* 10/7, 2020.
- [HJP22] Hoque, M. M.; Jakowski, N.; Prol, F. S.: A New Climatological Electron Density Model for Supporting Space Weather Services. *J. Space Weather Space Clim.* 12/1, 2022.
- [LH14] Liu, Z.; Heer, J.: The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graph.* 20/12, pp. 2122–2131, 2014.
- [SJH19] Su, K.; Jin, S.; Hoque, M. M.: Evaluation of Ionospheric Delay Effects on Multi-GNSS Positioning Performance. *Remote. Sens.* 11/2, p. 171, 2019.

# Interactive SQL Queries and Program Code in Presentations

Johannes Schildgen,<sup>1</sup> Florian Heinz<sup>1</sup>

**Abstract:** Nowadays, most database lectures are performed with an accompanying visual presentation that further illustrates the conveyed facts. Conventional presentation software allows dynamic elements up to a certain level, for example revealing or changing parts of the slide step by step, or even an interaction with the viewers by means of polls or similar mechanisms. Recently, HTML- and browser-based frameworks for presentations have emerged, which allow an even higher degree of flexibility due to the manifold possibilities of HTML5 and JavaScript. This paper presents an approach of how to interactively modify parts of a slide during the presentation, like SQL-based queries or program code snippets, and show the results pretty-printed on the corresponding slide in real-time. This enables the lecturer to easily show more examples, and answer and illustrate side questions, which they did not prepare in advance.

**Keywords:** Lecture Slides; SQL

## 1 Introduction

Today, a lecture is not thinkable without an accompanying presentation that helps to visually illustrate the topic. Often, this presentation is handed out to the audience and replaces, or at least supports, handwritten notices taken by the audience. Classical presentation software is, for example, Microsoft Powerpoint, Apple's Keynote, or LibreOffice Impress, which have roughly the same feature set available. Basic features are to reveal parts of a slide step-by-step or provide animations. To implement more advanced features for more flexibility in the presentation, software-specific plugins are needed here, for example, poll-plugins to interact with the audience.

Recently, another method to create presentations has become more and more popular: browser-based presentation frameworks that make heavy use of HTML5, CSS, and JavaScript. For example, one of such frameworks is reveal.js [EH13]. A big advantage is the comparatively easy extensibility of presentation features due to the open nature of the platforms used, to tailor the presentation to the specific needs of the lecture topic.

In this paper, we present an extension to reveal.js that can be used for lectures in computer science. Our first implementation was able to send SQL queries on the slides to a real database, executes them, and shows the results directly within the slides. Then, we extended our approach to supporting arbitrary program code.

---

<sup>1</sup> OTH Regensburg, Postfach 120327, 93025 Regensburg, Germany  
{johannes.schildgen,florian.heinz}@oth-regensburg.de

The traditional way of creating SQL slides is to think about an example, formulate a database query for it, execute it in a database system and copy both the query and its result into the presentation. The disadvantage of this approach becomes imminent when someone in the audience asks a *but what if?*-question, where the query is slightly altered. With the traditional tools, the lecturer has to stick with his prepared visualization and can only explain verbally, what will happen or they have to make a context switch and reproduce the query within an SQL client software.

This is where the new method described in this paper comes into play; a JavaScript-based plugin for reveal.js that executes a query shown in the presentation and presents the results to the audience. This enables the lecturer to flexibly react on questions regarding the queries that are raised by the audience. Further benefits of this feature are: No erroneous queries on the slides (it would show an error message), the avoidance of frequently switching between the presentation window and a console (and setting up an example environment), and consistent DB schemata and data values with less effort to produce, as they are based on a real relational database.

In programming lectures, frequently similar situations occur. Small changes to the code often result in completely different behavior. Operator precedence, lazy evaluation, out-of-order processing, and many things more can be illustrated swiftly if questions occur spontaneously. So, it is a big advantage if the code snippet is already part of a runnable program with the irrelevant parts hidden and its output presented on the slide: for one, there is no room for inconsistency or typos on the slide, because it is indeed compiled and executed. For another, there is no need for the lecturer to set up an IDE for the programming language in question and write boilerplate code to get a specific example running. This saves a lot of time and does not confuse the audience: The bigger context of the code part can be concealed, but modifications to the shown excerpt result in an immediate update of the output on the slide, together with possible compile-time and runtime errors.

## 2 Related Work

**reveal.js** The HTML5- and JavaScript-based framework reveal.js [EH13] is the foundation for the work presented here. Creating a presentation with reveal.js is similar to designing an HTML website. For example, each slide is one `<section>` element, and the sizes of a slide's contents are automatically adjusted to the size of the browser viewport. reveal.js offers a speakers view, math formulae, PDF export, syntax highlighting for code, slide transitions, and fragments, i.e., elements on a slide can be revealed step-by-step. One of the most useful features is the plugin interface<sup>2</sup>, which allows to extend the framework with self-programmed functions, as for example polls, diagram-creation functions, or dynamic SQL queries.

---

<sup>2</sup> <https://revealjs.com/plugins/>

**LaTeX Beamer** LaTeX Beamer is a popular alternative to PowerPoint and other presentation tools, especially in maths and science education. It works in a similar way as reveal.js: A text-based syntax is used to develop and style the content of the presentation slides, and then a PDF is generated, which can be presented to the audience with a PDF viewer. There are tools like SQLTeX [vE16] and LaTeXDB [Eß06], which act as a preprocessor for LaTeX files. They execute SQL queries that are part of the file on a relational database and write their results directly into the LaTeX document. This approach is similar to ours but—due to the output format PDF—it does not allow for interactively changing and re-executing the query while presenting the slides.

**Jupyter Notebook** Jupyter Notebooks [KI16, GG16, Pe18] are often used by data scientists to combine program code, documentation, and visualization within a file that can be inspected and executed within a browser. As this format is very interactive, users can change the program code and see its effects immediately. With plugins like sqlalchemy [Co08] or SQLFlow [Wa20], it is possible to include and execute SQL queries within a Jupyter Notebook. Another plugin, RISE [Av17], allows for presenting the notebook as a sequence of slides. While this approach follows the idea of creating an interactive document and presenting this in the form of slides, our approach is vice versa: We will create slides and enrich them with interactive elements.

**LiaScript** LiaScript [Di19] is a browser-based tool using an extended Markdown language to create interactive online courses. It is designed to be easy-to-use and to provide a high level of flexibility; the markdown source is rendered live in the browser itself. It also provides the ability to execute code snippets, perform quizzes, surveys and more. The focus is on providing an online course, while this work strives to augment a presentation with interactive elements, that is performed by a lecturer with a live audience.

### 3 Dynamic Content Evaluation in Presentations

A central element of a successful lecture is illustrating the theoretical concepts by means of examples. This helps to avoid misunderstandings and provokes questions from the public that otherwise would not have come up. Usually, the lecturer strives to find suitable examples and tries to anticipate possible questions to answer them in that course. This will, however, not be possible in all situations. So, a dynamic component is the best option to provide maximum flexibility during the presentation.

For example, the lecture comes to a point where NULL values in SQL are discussed and the lecturer tries to convey that each comparative relational operation, i.e. less than, greater, equality or inequality, with a NULL value is neither true, nor false, but NULL (“unknown”) in the result and the correct way to test for such values to use the IS NULL operation. Then, one slide might show a query `SELECT * FROM people WHERE email IS NULL` (see Figure 1).

# NULL Values

```
SELECT * FROM people WHERE email IS NULL
```

| name            | phone       | email |
|-----------------|-------------|-------|
| James Smith     | 1-555-31337 | -     |
| Anna Doe        | 1-555-12325 | -     |
| Frank Hollywood | 1-555-21312 | -     |

Fig. 1: Dynamic Query

Below the SQL statement, an exemplary result table with several rows that might be a possible result of that query is shown. This is the primarily relevant information and will also be part of the (printed or digital) handout. But when dynamic queries are used, some more possibilities exist to further clarify this fact. The query is editable and the IS operator can be replaced by the relational equality operation =. The keyboard shortcut `ctrl + ↵` re-executes that query live in an SQL database and shows that the result set is indeed empty. After that, the lecturer might be confronted with the question of what the result with the != operation might be (people tend to think, that might yield all rows from the table then). Again, the operator is quickly replaced and the audience notices, that the result set is empty again. The reason for this can then be reiterated from the theoretical explanation beforehand and the chances for a deeper understanding of the underlying mechanisms rise.

Another often-heard question when talking about the JOIN operator is: “How would the result look with a left join?”. Using this framework, this can also be demonstrated easily by changing the query in the slide accordingly. These are rather simple examples, but basically all kinds of queries are possible.

The more complex the query, the more nuances can be demonstrated by changing small parts of the query and explaining the observed changes in the result. This is exactly what the SQL plugin is designed for. Figure 2 shows how the plugin works. For the database system in the backend, there are two fundamentally different possibilities. For one, some modern web browsers (e.g., Chrome and Opera) contain an internal WebSQL API that allows to create and query a relational database from JavaScript without the need of any external program or service. This also means, that the whole presentation can be held from local files without the need of a web server. This is convenient, however, there are some drawbacks with this approach. The Web SQL API has been deprecated since 2010. This leads to the fact, that Mozilla Firefox and other major players in the browser business do not support this API. The browsers that do support it all use SQLite as implementation, which provides a rich set of SQL features. However, sometimes more complex database operations have to be

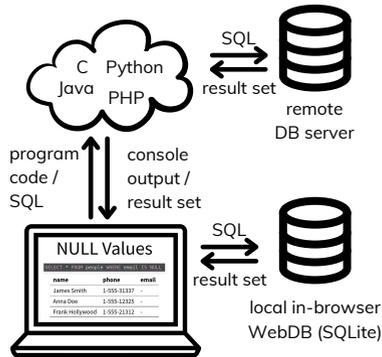


Fig. 2: Operating principle of the plugin

discussed, which might not be supported by SQLite, full outer joins and stored procedures for instance. For demonstrating code snippets in programming lectures, a backend server that compiles and executes code snippets is always needed.

To also be able to cover these and more topics within a database-systems lecture, this plugin also implements the possibility to connect to external database systems by proxying the query through a server-side PHP script, which constitutes the adapter between the dynamic JavaScript query functions and a database system, for example, a PostgreSQL server. Even NoSQL databases can be connected with some minor effort here. Either way, the user has the option to reset the database to a default state before loading the presentation or to stick with the current state, which might be preferable when methods for modifying the database scheme or the data itself are currently on-topic.

As already mentioned, the presentation framework used in this paper is reveal.js, which basically is a set of CSS and JavaScript files to set up a presentation by using HTML. The SQL query plugin uses the plugin interface of the framework and can easily be integrated by downloading our `sql.js` file (URL: see below) and adding it to the list of dependencies: `{ src: 'src/sql.js', async: true }` After that, dynamic queries can be defined in the presentation source like this:

```
<pre><code class="sql" contenteditable data-sql-engine="postgresql">
SELECT * FROM people WHERE email IS NULL</code></pre>
<span class="sqlresult"></span>
```

The result might be rendered as shown in Figure 1. If the query would contain an error, the message from the database would be presented to the audience in a popup window.

Within the `<code>` element, the following classes and attributes are supported by this implementation:

- `sql, java, . . .`: classes for syntax highlighting from the library `highlight.js`; furthermore, an indicator for our plugin to execute the query or code
- `contenteditable`: makes the code editable and re-executable
- `data-sql-engine`: allows for using different backend database connections
- `dont_execute_sql`: The SQL query should not be executed (for `INSERT, UPDATE, . . .`)
- `sqlresult`: class of `span` in which the result table will be shown
- `data-sql="some_id" / result="some_id"`: result table / program output will be shown in the `span` with the given id (if there are multiple code blocks on one slide)

The following will execute and show the results of a SQL query that is not shown on the slides: `<span data-sql-query="select * from tbl"></span>` This is useful for simply showing a full table, for showing results of relational-algebra expressions, or for faking queries (e.g., `A right join B` is shown on the slides, but `B left join A` is executed instead, because SQLite does not support right joins).

## 4 Conclusion

This paper presents a technique for creating slides for database lectures and other teaching presentations for computer science. SQL queries and program code are executed on a real system, the result is shown to the audience. This enables the lecturer to flexibly react to what-if questions from the audience without having to break the medium and frequently switch to a database client, an IDE, or similar.

While results of SQL queries are displayed in a table format, results of general program-code snippets is simple plain-text console output. By using JavaScript, this can be further processed to display, for example, charts, diagrams, chemical formulae, and much more.

The implementation of the dynamic SQL-query plugin for `reveal.js` can be found under <https://github.com/jschildgen/db-slides>.

## Description of the Demonstration

We will present the `reveal.js` presentation framework together with our extensions to the conference visitors in an interactive way. Everybody can click through slides, edit and execute SQL queries and code, and they can inspect the source code. Furthermore, we present more features and extensions of `reveal.js`, namely an ER-diagram creator and a poll plugin [Sc21]. The audience can use their own mobile phones to participate in polls and see how their voting will affect the presentation.

## Bibliography

- [Av17] Avila, Damian: Rise: Reveal.js-jupyter/ipython slideshow extension. 2017.
- [Co08] Copeland, Rick: Essential sqlalchemy. O'Reilly Media, Inc., 2008.
- [Di19] Dietrich, André: LiaScript: a domain-specific-language for interactive online courses. In: part of the Multi Conference on Computer Science and Information Systems 2019. p. 186, 2019.
- [EH13] El Hattab, Hakim: The HTML presentation framework. reveal.js, Jul 2013.
- [Eß06] Eßer, Hans-Georg: LaTeXDB - Integrates LaTeX and SQL databases. 2006.
- [GG16] Granger, B; Grout, J: JupyterLab: Building Blocks for Interactive Computing. Slides of presentation made at SciPy, 2016.
- [K116] Kluver, Thomas; Ragan-Kelley, Benjamin; Pérez, Fernando; Granger, Brian E; Bussonnier, Matthias; Frederic, Jonathan; Kelley, Kyle; Hamrick, Jessica B; Grout, Jason; Corlay, Sylvain et al.: Jupyter Notebooks-a publishing format for reproducible computational workflows. In: ELPUB. 2016.
- [Pe18] Perkel, Jeffrey M: Why Jupyter is data scientists' computational notebook of choice. Nature, 2018.
- [Sc21] Schildgen, Johannes: Interaktive Vorlesungsfolien mit SQL-Unterstützung. Datenbank-Spektrum, 21(1):19–27, Mar 2021.
- [vE16] van Eijk, Oscar: SQLTex v2.0. 2016.
- [Wa20] Wang, Yi; Yang, Yang; Zhu, Weiguo; Wu, Yi; Yan, Xu; Liu, Yongfeng; Wang, Yu; Xie, Liang; Gao, Ziyao; Zhu, Wenjing et al.: SQLFlow: A Bridge between SQL and Machine Learning. arXiv preprint arXiv:2001.06846, 2020.



## Workshop Track



## 2nd Workshop on Novel Data Management Ideas on Heterogeneous (Co-)Processors (NoDMC)

Dirk Habich,<sup>1</sup> David Broneske<sup>2</sup>

The objective of this one-day workshop is to explore the challenges and opportunities of data processing on existing and future heterogeneous hardware architectures. On the one hand, today's processors are no longer mainly bound by the density and frequency of transistors, but by their power and heat budgets. The so-called "power wall" forces hardware suppliers to rely more on the design of specialized devices optimized for certain types of calculations, which results in an increasingly heterogeneous processor landscape. On the other hand, memory and storage has seen an unprecedented change as well: novel and already commercially available techniques have blurred the traditional mental picture of a memory/storage hierarchy. For example, Non-Volatile RAM (NVRAM) is a prominent example to question the long-standing memory hierarchy reflected in almost all system-level applications. Moreover, very large caches, High-Bandwidth-Memory (HBM), Non-Uniform Memory Access (NUMA), or even remote-memory designs as well as extremely fast SSDs add to the heterogeneous portfolio of available memory/storage techniques. Therefore, to meet the performance requirements of the modern information society, tomorrow's database systems will have to exploit and embrace this increased heterogeneity of processor and memory technologies.

The purpose of this workshop is to assist with training and fostering a community of researchers and industry practitioners working on data processing issues on heterogeneous hardware systems. To this end, we want to provide a forum to discuss challenges, progress and directions, and to offer an environment for networking persons researching on related topics and fostering future collaborations. Especially in the view of the SPP 2037 on *Scalable Data Management for Future Hardware* and the SPP 2377 on *Disruptive Memory Technologies*, we want to strengthen collaborations beyond individual SPP projects by connecting them with other researchers. This workshop is co-organized by the GI-Arbeitskreis *Data Management on Modern Hardware*.

The scope of the workshop includes, but is not limited to:

---

<sup>1</sup> TU Dresden, dirk.habich@tu-dresden.de

<sup>2</sup> German Centre for Higher Education Research and Science Studies, broneske@dzhw.eu

- Applications of modern hardware in
  - data mining
  - data-intensive machine learning
  - query processing
  - sensor or stream processing
  - non-traditional applications (e.g., graph processing)
- Algorithms and data structures for efficient data processing on and across different (co-)processors or memory technologies
- Exploitation of specialized ASICs or specialized memories technologies (e.g., processing in memory (PIM))
- Efficient memory management, data placement and data transfer strategies in heterogeneous systems
- Energy efficiency in heterogeneous hardware environments
- Programming models and hardware abstraction mechanisms for writing data-intensive algorithms on heterogeneous hardware
- Query optimization, cost estimation and operator placement strategies for heterogeneous hardware
- Transaction processing in heterogeneous systems

With the given scope of the workshop, we are happy to announce a great program. The workshop starts with a keynote by David F. Bacon working at Google Research, who is the leading architect of the Spanner storage engine. From the submissions, we were able to accept five technical papers as well as four extended abstracts. The corresponding talks are organized in two sessions according to the topics of *Advances in Storage, Memory, and Network Technologies* in Session 1 and *Advances in (Co-)Processing Technologies* in Session 2. The first talk in the first session is by Baumstark et al. who investigate the capabilities of processing-in-memory technologies for table scans. Afterward, El-Shaikh et al. present how to store information using DNA-based storage systems. The third technical paper by Lutsch et al. focuses on the performance of SGX for machine learning workloads. Next, the extended abstract of Benson et al. gives lessons learned of using persistent memory under CXL. The last talk in this session by Geyer et al. discusses benefits and drawbacks of CXL for heterogeneous cloud architectures.

In Session 2, four papers in the area of (co-)processor acceleration are presented. The first talk by Damme and Boehm is an extended abstract to a CIDR paper presenting an architecture for exploiting heterogeneous processors for data science applications. Afterward, Hahn et al. present an FPGA parser and an according parser generator for the Apache Avro

format – a semi-structured data format used in stream processing applications. The third talk is by Schuhknecht and Islam, who benchmark heterogeneous multi-core CPUs running multiple queries at a time in parallel. The workshop program closes with the presentation by Fett et al. who investigate the performance of matrix multiplication under different compressed formats and overflow handling on GPUs.

Last but not least, we like to thank everyone who contributed to this workshop, in particular, the authors, the reviewers, the BTW team, and all participants.

## **PC Chairs**

- David Broneske (DZHW)
- Dirk Habich (TU Dresden)

## **Steering Committee**

- Wolfgang Lehner (TU Dresden)
- Gunter Saake (University of Magdeburg)
- Kai-Uwe Sattler (TU Ilmenau)

## **Program Committee**

- Sebastian Breß (Snowflake)
- David Broneske (DZHW)
- Patrick Damme (TU Berlin)
- Philipp Götze (SAP SE)
- Dirk Habich (TU Dresden)
- Tilmann Rabl (HPI Potsdam)
- Hannes Rauhe (SAP SE)
- Horst Schirmeier (TU Dresden)
- Knut Stolze (IBM Germany)
- Annett Ungethüm (Universität Hamburg)
- Stefan Wildermann (Friedrich-Alexander Universität Erlangen-Nürnberg)
- Steffen Zeuch (DFKI und TU Berlin)



## Fourth Workshop on Big (and Small) Data in Science and Humanities (BigDS 2023)

Andreas Henrich,<sup>1</sup> Naouel Karam,<sup>2</sup> Birgitta König-Ries,<sup>3</sup> Bernhard Seeger<sup>4</sup>

In the last 20 years, we have seen a continuous digital transformation in science, society, and economy. The growth of the internet and advancements in data collection have resulted in the era of Big Data, marked by a massive and continually growing amount of complex, interconnected, and heterogeneous data. Earth observation sensors, for instance, produce petabytes of data with improved spectral, temporal, and spatial precision. Social media users generate high volumes of content. The information and knowledge contained in these data have huge potential value, which, if uncovered, could aid in improving our understanding of complex systems such as earth and society, drive innovation, and empower well-informed decisions.

Thus, the importance of data has increased dramatically not only in business, but also in almost all scientific disciplines, e.g., in meteorology, genomics, complex physical simulations, bio- and environmental research, and more recently in the humanities. This led in particular to the creation of the unique NFDI (National Research Data Infrastructure), which aims to “systematically manage scientific and research data, provide long-term data storage, backup and accessibility, and network the data both nationally and internationally”<sup>5</sup>. NFDI began with more than 25 domain-specific consortia and projects in the area of basic infrastructure, covering a broad range of scientific disciplines from cultural sciences, humanities and engineering to life and earth sciences.

The availability of such a large volume of multidisciplinary data within NFDI and beyond leads to a rethinking in scientific disciplines on how to extract relevant information and on how to foster research. Researchers face severe challenges in leveraging data, since appropriate data management, integration, analysis and visualization tools have not been available so far. Recent advances in the development of big data technologies and the progress in machine learning and semantic technologies allow for a better computational support to deal with large amounts of heterogeneous data, and offer flexible end-to-end analytic and

---

<sup>1</sup> University of Bamberg, Media Informatics, 96047 Bamberg, Germany [andreas.henrich@uni-bamberg.de](mailto:andreas.henrich@uni-bamberg.de)

<sup>2</sup> Fraunhofer FOKUS & InfAI e.V., Berlin, Germany [karam@infai.org](mailto:karam@infai.org)

<sup>3</sup> University of Jena, Heinz Nixdorf Chair for Distributed Information Systems, 07743 Jena, Germany [birgitta.koenig-ries@uni-jena.de](mailto:birgitta.koenig-ries@uni-jena.de)

<sup>4</sup> University of Marburg, Department of Mathematics and Computer Science, 35032 Marburg, Germany [seeger@informatik.uni-marburg.de](mailto:seeger@informatik.uni-marburg.de)

<sup>5</sup> [https://www.dfg.de/en/research\\_funding/programmes/nfdi/index.html](https://www.dfg.de/en/research_funding/programmes/nfdi/index.html)

visualization solutions for various application domains. A critical prerequisite for achieving those goals is the availability of data that is harmonized and made reusable in a sustainable and qualitative manner. This needs to be realized following the FAIR data principles, the fundamental concepts that aim to improve findability, accessibility, interoperability, and reusability of research data<sup>6</sup>.

The need to discuss real-world problems in data science as well as the recent advances in big data technology between database researchers and scientists from various disciplines already led to the first three editions of the workshop on Big (and Small) Data in Science and Humanities (BigDS) at BTW 2015, 2017, and 2019. This year's fourth edition of the BigDS workshop co-located with the 20th Conference on Database Systems for Business, Technology and Web (BTW) accommodates the continuously growing interest in methods to efficiently and effectively manage and analyze Big Data. With workshop contributions from various disciplines, we hope to promote the dialog between domain experts and data scientists and to foster the engagement of the database community to NFDI and other important infrastructure projects.

The workshop program kicked off with Markus Stocker, who gave an inspiring keynote on machine actionable scientific information. He introduced the basic concepts, discussed the challenges, and pointed out the great opportunities for producing and sharing knowledge in research and society.

We further selected eight contributions that address different challenges in the context of data-driven processing and analytics. The papers contribute to broadly applicable technologies like provenance for spreadsheets, management and integration of geo-spatial data, ontologies, trust in AI, and user interfaces. The proposed approaches are applicable to various domains, such as ecology and digital humanities.

Two papers focus on basic methods and systems for data processing. Müller and Mertová addressed the provenance problem of data transformations in spreadsheets. Their approach creates a copy of the source data in a new worksheet and performs data transformations on this copy while referring back to the original sheet. Beilschmidt et al. presented the basic concepts of Geo Engine, a new spatio-temporal processing infrastructure that has been used in several ecological projects, including NFDI4Biodiversity. Their workflows with a spatio-temporal context offer great potential and flexibility for many applications.

There are two papers addressing data extraction and data integration in scientific applications. Bartsch et al. described an extraction process of various digital objects from different sources to create a multimodal corpus for the analysis of climate change publications. Using a variety of tools, they manage to extract images, graphs, tables or videos and annotate text. Jegan et al. described an approach to support information integration and improving the data quality by using multiple external information sources to facilitate disambiguation of geographic data. The resulting system will be used within the infrastructure of the NFDI project Text+.

---

<sup>6</sup> <https://www.go-fair.org/fair-principles/>

As more data is available, dataset discovery is a frequent task in daily research practice. Thus, the question of user interfaces becomes more important that is addressed in the following two contributions. Löffler et al. proposed a semantic search for biological datasets. The authors evaluated two kinds of search interfaces including free text, categories and annotating of returned research results. Their results show that users prefer interfaces with a single input field for search tasks and appreciate explanations of the results. Schildgen et al. reported on an Alexa-based NLP tool to facilitate natural language querying of databases using SQL. This also includes the translation of the query result (which is always a table) into text and voice. Such kind of easy-to-use interfaces would widely facilitate the interaction with scientific databases.

The work of Abdelmageed et al. addressed the problem of knowledge transfer on ontologies and data integration towards a concrete application. The authors developed an agricultural core ontology that is used to link general concepts to more domain-specific concepts. Bruchhaus et al. presented how trust can be introduced into big data analysis and AI. Their prototype offers a so-called trust-bus as a component in a microservice architecture.

All contributions to this year's BigDS workshop provide new domain-relevant insights and promote the use of generic as well as domain-specific methods for scientific data management and analysis. We want to thank everyone who contributed to the workshop, especially the authors, the keynote speaker Markus Stocker, the BigDS program committee, the BTW team, and all the participants. We are grateful to NFDI4Biodiversity for its financial support of the workshop.

## **Workshop Organizers**

Andreas Henrich (Univ. Bamberg)  
Naouel Karam (Fraunhofer FOKUS & InfAI e.V.)  
Birgitta König-Ries (Univ. Jena)  
Bernhard Seeger (Univ. Marburg)

## **Program Committee**

Alsayed Algergawy (Univ. Jena)  
Thomas Brinkhoff (FH Oldenburg)  
Michael Diepenbroek (GFBio e. V., Bremen)  
Jana Diesner (University of Illinois at Urbana-Champaign)  
Michael Gertz (Univ. Heidelberg)  
Anika Groß (Hochschule Anhalt)  
Anton Güntsch (Botanischer Garten und Botanisches Museum, Berlin)  
Dominik Hezel (Univ. Frankfurt)

Alfons Kemper (TU München)  
Toralf Kirsten (Univ. Leipzig)  
Meike Klettke (Univ. Regensburg)  
Ulf Leser (HU Berlin)  
Richard Lenz (Univ. Erlangen)  
Ulrike Lucke (Univ. Potsdam)  
Bertram Ludäscher (University of Illinois at Urbana-Champaign)  
Manja Marz (Univ. Jena)  
Wolfgang Müller (HITS, Heidelberg)  
Thorsten Papenbrock (Univ. Marburg)  
Kai-Uwe Sattler (TU Ilmenau)  
Sirko Schindler (DLR Jena)  
Heiko Schuldt (Univ. Basel)  
Uta Störl (Fernuni Hagen)  
Dagmar Triebel (SNSB, München)  
Matthias Weidlich (HU Berlin)  
Claus Weiland (Senckenberg Gesellschaft für Naturforschung, Frankfurt)

# Workshop on Data Engineering for Data Science

Ralf Schenkel<sup>1</sup>, Ansgar Scherp<sup>2</sup>

## 1 Overview

Data engineering is a crucial part of any data science project: Data collection and metadata management are the prerequisite of any meaningful analysis and, in practice, take up the bulk of time spent in data science projects. These aspects, however, are often neglected in research in favor of more mathematical aspects. The workshop thus focused on typical data engineering topics such as data preparation and integration, scalable processing of data science processes, data quality, and benchmarks. In addition to regular papers, the workshop also offered to submit short reports on work in progress, applications, and tools (e.g., interesting use cases, problems, data sets, benchmarks, visionary ideas, system designs, and descriptions of system components and tools). Also, relevant papers that were already accepted at major database conferences or journals could be presented.

The workshop was an initiative of the DBIS working group „Data Engineering for Data Science“.

## 2 Program

The workshop featured an opening keynote by Markus Stocker (TIB Hannover) on recent advances in the Open Research Knowledge Graph, jointly with the workshop on Big (and Small) Data in Science and Humanities.

The first session focused on maintaining provenance information, which is clearly an important aspect of data engineering pipelines. In the first paper by Erik Kleinstauber et al. (University of Jena), a provenance management framework for knowledge graph generation was presented. The second paper by Dominik Kerzel et al. (University of Jena) introduced provenance management for data science notebooks. The last paper of this session by Maximilian Emanuel Schüle et al. (University of Bamberg and TU Munich) presented novel approaches for recursive SQL and GPU support for in-database machine learning.

The second session started with two papers on fairness and responsibility in data science applications. The first paper by Sabrina Göllner and Marina Tropmann-Frick (Hamburg

---

<sup>1</sup> Universität Trier, schenkel@uni-trier.de

<sup>2</sup> Universität Ulm, ansgar.scherp@uni-ulm.de

University of Applied Sciences) presented VERIFAI, aiming at evaluating the responsibility of AI systems. The second paper by Valerie Restat et al. (University of Hagen and University of Regensburg) introduced the design of a framework of metrics that allows for a flexible evaluation of data quality and data preparation results. The third paper by Arne Grünhagen et al. (HAW Hamburg, Hamburg University of Technology and Deutsches Elektronen-Synchrotron) included the results of a systematic literature study on predictive maintenance for optical synchronization systems.

The third and last session featured four papers. The first paper by Pronaya Prosun Das (Fraunhofer ITEM) discussed a solution for associative clustering in pediatric intensive care. The second paper by Björn Engemann and Philipp Schaer (Cologne University of Applied Sciences) introduced reliable rules for relation extraction in a multimodal setting. The third paper by David Burrell et al. (TU Berlin, DFKI, and IT University of Copenhagen) focused on workload prediction for IoT data management systems. The fourth and last paper in this session by Sven Langenecker et al. (DHBW Mosbach and TU Darmstadt) presented a new corpus for semantic type detection.

# A Tutorial Workshop on ML for Systems and Systems for ML

Manisha Luthra,<sup>1</sup> Andreas Kipf,<sup>2</sup> Matthias Boehm<sup>3</sup>

**Abstract:** This tutorial workshop on ML for Systems and Systems for ML is held on Tuesday, March 7th in conjunction with BTW 2023. In this workshop, we invite and bring together researchers working actively in the research areas at the intersection of machine learning and data management systems. There are invited/nominated talks on two topics of concern: (1) how machine learning can help or aid in data management system tasks (ML for Systems side) and (2) how scalable and efficient system design can improve machine learning pipelines (Systems for ML side). The talks will present accepted peer-reviewed work in a tutorial fashion to give hints on the current establishment in the two topics of concern and open research challenges thereby. The workshop showcases 13 high-quality talks with speakers from North America and all over Europe.

## 1 Introduction

The current advances in machine learning (ML) have led to a wide adoption of ML in different application areas across academia as well as industry. On the one hand, these novel advancements have helped existing data management systems to improve, by sometimes even completely replacing specific components with so-called *learned system components* (ML for Systems area). On the other hand, the advancements in well-thought and engineered systems aid in improvements of current ML techniques (Systems for ML area). For instance, in databases, there has been a surge in replacing the manually designed parts of databases with learned counterparts, such as learned query optimizers, learned indexes, learned cardinality and cost estimators, and even query schedulers. There have been also prominent examples where data access methods, such as indexing, and data flow optimizations have improved ML techniques. Such approaches have led to autonomy in developing data management systems and hence avoiding manual tuning by an administrator that current data management systems succumb in.

It is often challenging for researchers to keep up with the pace of these two emerging research areas, which is what we aim to make easier by means of this workshop tutorial. Therefore, in this workshop, we invite 13 speakers working in these areas who will present their already accepted peer-reviewed work in a tutorial fashion and give hints on their current work and open research challenges they are currently facing.

---

<sup>1</sup> TU Darmstadt and DFKI, manisha.luthra@dfki.de

<sup>2</sup> Amazon Web Services

<sup>3</sup> TU Berlin, matthias.boehm@tu-berlin.de

## 2 Organization Committee

**Workshop Co-chairs.** This workshop is co-organized by the following workshop chairs drawing from their previous experience and papers in this area.

- Manisha Luthra (TU Darmstadt)
- Andreas Kipf (Amazon Web Services)
- Matthias Boehm (TU Berlin)

**Local Organization Chair.** The local organization of the workshop as well as the website is handled by Lucas Woltmann from TU Dresden.

## 3 Workshop Format

**Hybrid format.** While we encourage in-person speakers and participants, we allow for hybrid attendance to account for the current travel restrictions. Therefore, the workshop is held in a hybrid format (in-person and remotely) with the following full-day schedule.

- The first half covers the Systems for ML area
- The second half covers the ML for Systems area

**Invited Speakers.** We have invited a balanced mix of speakers presenting their published prior work and open challenges in these two areas. With this workshop, we want to foster discussions and collaborations among the participants, and at the same time, give the speakers a platform to boost their work and gain visibility.

- Ziawasch Abedjan (Leibniz University Hannover)
- Stefan Hagedorn (TU Ilmenau)
- Benjamin Hilprecht (TU Darmstadt)
- Madelon Hulsebos (University of Amsterdam)
- Ryan Marcus (University of Pennsylvania)
- Arnab Phani (TU Berlin)
- Theodore Rekatsinas (ETH Zurich)
- Alexander Renz-Wieland (TU Berlin)
- Sebastian Schelter (University of Amsterdam)
- Stefanie Scherzinger (University of Passau)
- Maximilian E. Schüle (University of Bamberg)
- Immanuel Trummer (Cornell University)
- Giorgio Vinciguerra (University of Pisa)

**Workshop on Novel Data Management Ideas on  
Heterogeneous Hardware Architectures (NoDMC)**



# Benchmarking the Second Generation of Intel SGX for Machine Learning Workloads

Adrian Lutsch,<sup>1</sup> Gagandeep Singh,<sup>1</sup> Martin Mundt<sup>1,2</sup>, Ragnar Mogk,<sup>1</sup> Carsten Binnig<sup>1</sup>

**Abstract:** For domains with high data privacy and protection demands, such as health care and finance, outsourcing machine learning tasks often requires additional security measures. Trusted Execution Environments like Intel SGX are a powerful tool to achieve this additional security. Until recently, Intel SGX incurred high performance costs, mainly because it was severely limited in terms of available memory and CPUs. With the second generation of SGX, Intel alleviates these problems. Therefore, we revisit previous use cases for ML secured by SGX and show initial results of a performance study for ML workloads on SGXv2.

**Keywords:** Trusted Execution Environments; Intel SGX; Machine Learning; Benchmarking

## 1 Introduction

**The Importance of Trusted Computing.** Trusted Execution Environments (TEE) are a powerful tool for privacy-preserving, trusted, and secure data processing in cloud environments. TEEs have been used to build secure systems like databases [PVC18; VGG19], storage engines [Su21], and data processing systems [Sc15]. Furthermore, they have also been used for secure machine learning (ML) systems. This includes work for secure neural network training [Hu18; Le20; Oh16; Qu20] or secure inference [Hu18; Le19; Qu20], secure federated learning [KCZ21; Mo21a; QF21; Qu20], and many more.

**Intel SGX for Trusted Computing.** The most widely used TEE implementation of the aforementioned systems is Intel Software Guard Extensions (SGX) [An13; CD16; In22b; Mc13]. Intel SGX assures the integrity of processes and the confidentiality of their data by running them inside of protected memory regions called enclaves. Data inside an enclave can only be accessed by the process running inside the same enclave, not by other processes, the operating system, or a hypervisor. Additionally, SGX supports so-called attestation. With attestation, a process can prove that it is running the expected code inside an enclave [An13; CD16; Mc13]. Thereby, SGX protects against strong adversaries with full control over operating system and hardware.

**SGXv2 relieves previous Limitations.** Although Intel SGX is a useful security technology, its first version (which we call SGXv1 in this paper) has severe limitations for the shielded applications, especially: (1) The encrypted and integrity-protected memory (called Enclave Page Cache, EPC) is limited to 128 MB, of which only ~90 MB are usable for user enclaves.

---

<sup>1</sup> Technical University of Darmstadt, Contact: [adrian.lutsch@cs.tu-darmstadt.de](mailto:adrian.lutsch@cs.tu-darmstadt.de)

<sup>2</sup> Hessian Center for AI ([hessian.AI](http://hessian.ai))

(2) Context switches between normal unprotected execution and secure protected execution of the enclave are costly. (3) Memory decryption and integrity protection cause overhead on cache misses, and (4) server-grade and multi-socket CPUs are not supported. With the most recent generation of SGX-enabled processors (which we call Second Generation of SGX or SGXv2 in this paper), Intel introduced several enhancements to the SGX technology which address some of the previous limitations [Jo21]. Primarily, new CPUs support up to 512 GB of EPC per CPU socket, which alleviates the need for expensive EPC paging. Additionally, SGX is now supported for multi-socket server systems. Enclaves on these systems can use the combined cores and EPC of all sockets, enabling higher degrees of parallelization.

**Revisit Secure ML on SGXv2.** These developments raise the question whether previous workarounds and optimizations for ML use cases in the restricted SGXv1 are still necessary. Therefore, we study the performance of ML use cases secured using SGXv2. Towards this goal, we measure the overhead of securely running ML tasks in SGXv2 enclaves and compare the results to SGXv1. The impact of SGXv2 was already analyzed for database workloads [E122]. However, we think that a closer look at machine learning workloads is justified because they have very different characteristics in terms of data access, compute intensity, and communication patterns. Furthermore, while the existing benchmarks are rather low-level, we investigate the performance of more complex algorithms and systems. In the rest of this paper we will present two of the most important use cases for Intel SGX in machine learning, Outsourced ML and Federated Learning, and report on our first evaluation results for outsourced neural network inference.

## 2 Using SGX for Secure ML

Since SGX is a versatile security technology with strong guarantees, it has been used to secure different applications in various settings. Next, we discuss two of the main use cases for SGX in secure ML and how SGXv1 limited them in terms of performance.

**Outsourced ML** is a setting in which an untrusted cloud provider offers infrastructure or services used for machine learning applications. In this setting, a malicious cloud provider is able to access the cloud customer's model and data while it is decrypted in memory. Furthermore, the cloud provider can also use its privileges to manipulate model and data causing false or low quality model predictions. TEEs have been shown to prevent such attacks [Gr19; Hu18; Le19; Le20; Oh16; Qu20; TB19]. For example, due to the confidentiality guarantees, cloud customers can be sure that the model can not be accessed by the cloud provider. Moreover, attestation and integrity guarantees ensure that the cloud provider does not tamper with the model without the customer being able to detect it.

The main limiting factor, however, of SGXv1 for this use case is the enclave memory size [Gr19; Le19; Qu20] which causes enclave paging. Enclave paging happens when the CPU-supported EPC is exceeded. Since today's deep neural network architectures commonly require gigabytes of memory and SGXv1 only supports 90 MB of active memory, previous approaches try to reduce memory consumption to prevent enclave paging [Le19; Qu20]. Furthermore, previous works suggest that parallelizing training and inference in

SGXv1 did not yield expected speedups [Qu20]. In Sect. 3 we report our results of using SGXv2 which shows that these limitations do not hold anymore.

**Federated Learning** is an approach for machine learning over data of multiple data owners. Instead of centralizing the data, the data owners (called clients) train their model together. In the centralized setup, each client trains the model on its own data and a central parameter server regularly collects model updates, averages them, and sends the updated model to all clients. This process is repeated until convergence. At the end of this process, all participants have a model trained on their joint data without exchanging the data itself [Li20]. Although Federated Learning can mitigate some privacy risks in machine learning by not centralizing the data, there exist attacks against it. For example, it has been shown that a curious parameter server can reconstruct training data from model updates of the clients [Ge20; Ph18]. Furthermore, a malicious parameter server can manipulate the model and the training process [Ge20]. Running the server inside an SGX enclave mitigates these kinds of attacks [KCZ21; Mo21a; Mo21b; QF21; Xu21].

The main limitation of SGXv1 for this use case is again the EPC size [KCZ21; Mo21a]. Additionally, the communication of clients and server via the network requires enclave transitions. Frequent enclave transitions are known as a bottleneck of Intel SGX. Each transition causes a constant overhead for flushing caches and TLBs as well as a linear overhead with the parameters and gradients copied to and from the encrypted memory region. Therefore, we will investigate how parameter servers for federated learning behave when secured by SGX enclaves and whether SGXv2 improves compared to SGXv1.

### 3 Benchmarking SGX Neural Network Inference

In this section, we report on our initial results depicted in Fig. 1 and 2. We analyze the overhead SGXv1 and SGXv2 cause when executing inference on neural networks of different sizes (Fig. 1) and investigate the potential speedup through parallelism enabled by more CPU cores in SGXv2 (Fig. 2). To show the influence of increasing network sizes, we compare a small multi-layer perceptron (MLP), a simplified version of AlexNet [KSH17], and the VGG19 convolutional neural network architecture [SZ15]. The MLP has three layers with sizes 784, 100, and 10. The AlexNet was simplified by replacing the three final layers with one layer of size 500. These networks thus cover a wide spectrum of model sizes: 2 MB (MLP), 80 MB (AlexNet) and 1.4 GB (VGG19). As such, VGG19 (1.4 GB) cannot be stored in the EPC of SGXv1 whereas the other models fit in the EPC of SGXv1. For our experiments, we use the Intel DNNL library available as part of the SGX SDK [In22a]. To show differences between SGXv1 and SGXv2, we executed the inference on an Intel Xeon E-2288G (SGXv1) and a server with two Intel Xeon Gold 6326 CPUs (SGXv2).

Fig. 1 shows the relative overhead ; i.e., the time required for inference inside an enclave divided by the time required without SGX on the same hardware. We can see that inference inside SGX has very low overheads if the enclave fits into the EPC and context switches are amortized. The overhead for the small MLP can be explained largely by the necessary

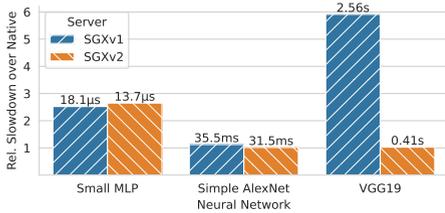


Fig. 1: Relative slowdown over native execution (bars) and absolute SGX times (labels).

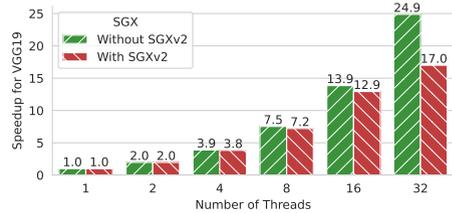


Fig. 2: Speedup by using multi-threading to parallelize ML inference with and without SGXv2.

context switches. Context switches are needed to copy the input data to the model in the enclave and inference results out of the enclave. Since the inference of the MLP takes only 5 - 7 microseconds, the relative cost of two context switches (that take  $\sim 4$  microseconds) is high. For the two larger neural networks, the relative context switch cost is negligible. For the AlexNet we measured around 10% slower inference for SGXv1 and only 3% slower inference for SGXv2. The large VGG19, which does not fit into the EPC of SGXv1, has a nearly 5 times longer inference time in SGXv1 due to EPC paging. On SGXv2, paging is not necessary, which leads to negligible overheads.

Additionally, in a second experiment we analyzed if the increased number of CPU cores of the SGXv2 hardware can be used to speed up inference of large networks that fit into the enlarged EPC. For the smaller models that would also fit into the EPC of SGXv1, the parallelization speed up with higher core counts is outweighed by the overhead of thread synchronization. Hence, we do not show these results. Fig. 2 shows the speedup gained through parallelization for the large VGG19 network on the SGXv2 hardware. When using only CPU cores on one socket, speedups with and without SGX are very similar. For example, with 16 threads we observe a 12.9 times speedup in an SGXv2 enclave and a 13.9 times speedup without. However, when the work is distributed over both sockets of the server, SGX seems to reduce parallelization gains. Using the 32 cores of both CPUs, we observe a 24.9 times speedup without SGX and only a 17 times speedup with SGXv2. We hypothesize that this is due to the non-uniform memory access and the encryption of communication between both CPUs in SGX mode.

## 4 Conclusion

We benchmark SGXv2 for ML workloads. Our first experiments show, among other insights, that the increased EPC capacity enables inference of today's deep neural networks with negligible overhead. We will continue our work with more in-depth analysis of neural network inference, other secure ML use cases, such as training and federated learning, an investigation into library operating systems like Gramine [Th22; TPV17], and an investigation into application optimizations for the new hardware.

## References

- [An13] Anati, I.; Gueron, S.; Johnson, S. P.; Scarlata, V. R.: Innovative Technology for CPU Based Attestation and Sealing, 2013, URL: <https://www.intel.com/content/dam/develop/external/us/en/documents/hasp-2013-innovative-technology-for-attestation-and-sealing.pdf>, visited on: 11/23/2022.
- [CD16] Costan, V.; Devadas, S.: Intel SGX Explained, 2016, URL: <https://eprint.iacr.org/2016/086.pdf>, visited on: 11/23/2022.
- [El22] El-Hindi, M.; Ziegler, T.; Heinrich, M.; Lutsch, A.; Zhao, Z.; Binnig, C.: Benchmarking the Second Generation of Intel SGX Hardware. In: Data Management on New Hardware. DaMoN'22, Association for Computing Machinery, New York, NY, USA, pp. 1–8, June 12, 2022, ISBN: 978-1-4503-9378-2.
- [Ge20] Geiping, J.; Bauermeister, H.; Dröge, H.; Moeller, M.: Inverting Gradients - How Easy Is It to Break Privacy in Federated Learning? In: Advances in Neural Information Processing Systems. Vol. 33, Curran Associates, Inc., pp. 16937–16947, 2020, URL: <https://proceedings.neurips.cc/paper/2020/hash/c4ede56bbd98819ae6112b20ac6bf145-Abstract.html>, visited on: 11/28/2022.
- [Gr19] Grover, K.; Tople, S.; Shinde, S.; Bhagwan, R.; Ramjee, R.: Privado: Practical and Secure DNN Inference with Enclaves, Sept. 5, 2019, arXiv: 1810.00602 [cs], URL: <http://arxiv.org/abs/1810.00602>, visited on: 04/08/2022.
- [Hu18] Hunt, T.; Song, C.; Shokri, R.; Shmatikov, V.; Witchel, E.: Chiron: Privacy-preserving Machine Learning as a Service, Mar. 15, 2018, arXiv: 1803.05961 [cs], URL: <http://arxiv.org/abs/1803.05961>, visited on: 04/07/2022.
- [In22a] Intel Corporation: Intel(R) Software Guard Extensions for Linux\* OS, Nov. 26, 2022, URL: <https://github.com/intel/linux-sgx/tree/master/external/dnnl>, visited on: 11/29/2022.
- [In22b] Intel Corporation: Intel® Software Guard Extensions, Intel, 2022, URL: <https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html>, visited on: 11/23/2022.
- [Jo21] Johnson, S.; Makaram, R.; Santoni, A.; Scarlata, V.: Supporting Intel SGX on Multi-Socket Platforms, 2021, URL: <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/supporting-intel-sgx-on-multi-socket-platforms.pdf>, visited on: 11/23/2022.
- [KCZ21] Kuznetsov, E.; Chen, Y.; Zhao, M.: SecureFL: Privacy Preserving Federated Learning with SGX and TrustZone. In: 2021 IEEE/ACM Symposium on Edge Computing (SEC). Pp. 55–67, Dec. 2021.
- [KSH17] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM 60/6, pp. 84–90, May 24, 2017, ISSN: 0001-0782.

- [Le19] Lee, T.; Lin, Z.; Pushp, S.; Li, C.; Liu, Y.; Lee, Y.; Xu, F.; Xu, C.; Zhang, L.; Song, J.: Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In: The 25th Annual International Conference on Mobile Computing and Networking. MobiCom '19, Association for Computing Machinery, New York, NY, USA, pp. 1–17, Oct. 11, 2019, ISBN: 978-1-4503-6169-9.
- [Le20] Lee, D.; Kuvaiskii, D.; Vahldiek-Oberwagner, A.; Vij, M.: Privacy-Preserving Machine Learning in Untrusted Clouds Made Simple, Sept. 9, 2020, arXiv: 2009.04390 [cs].
- [Li20] Li, T.; Sahu, A. K.; Talwalkar, A.; Smith, V.: Federated Learning: Challenges, Methods, and Future Directions. IEEE Signal Processing Magazine 37/3, pp. 50–60, May 2020, ISSN: 1558-0792.
- [Mc13] McKeen, F.; Alexandrovich, I.; Berenzon, A.; Rozas, C. V.; Shafi, H.; Shanbhogue, V.; Savagaonkar, U. R.: Innovative Instructions and Software Model for Isolated Execution. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. HASP '13, Association for Computing Machinery, New York, NY, USA, p. 1, June 23, 2013, ISBN: 978-1-4503-2118-1.
- [Mo21a] Mo, F.; Haddadi, H.; Katevas, K.; Marin, E.; Perino, D.; Kourtellis, N.: PPFL: Privacy-Preserving Federated Learning with Trusted Execution Environments. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. MobiSys '21, Association for Computing Machinery, New York, NY, USA, pp. 94–108, June 24, 2021, ISBN: 978-1-4503-8443-8.
- [Mo21b] Mondal, A.; More, Y.; Rooparaghunath, R. H.; Gupta, D.: Poster: FLATEE: Federated Learning Across Trusted Execution Environments. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). Pp. 707–709, Sept. 2021.
- [Oh16] Ohrimenko, O.; Schuster, F.; Fournet, C.; Mehta, A.; Nowozin, S.; Vaswani, K.; Costa, M.: Oblivious Multi-Party Machine Learning on Trusted Processors. In: 25th USENIX Security Symposium (USENIX Security 16). Pp. 619–636, 2016, ISBN: 978-1-931971-32-4, URL: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko>, visited on: 12/01/2022.
- [Ph18] Phong, L. T.; Aono, Y.; Hayashi, T.; Wang, L.; Moriai, S.: Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. IEEE Transactions on Information Forensics and Security 13/5, pp. 1333–1345, May 2018, ISSN: 1556-6021.
- [PVC18] Priebe, C.; Vaswani, K.; Costa, M.: EnclaveDB: A Secure Database Using SGX. In: 2018 IEEE Symposium on Security and Privacy (SP). 2018 IEEE Symposium on Security and Privacy (SP). Pp. 264–278, May 2018.
- [QF21] Quoc, D. L.; Fetzer, C.: SecFL: Confidential Federated Learning Using TEEs, Oct. 7, 2021, arXiv: 2110.00981 [cs].

- [Qu20] Quoc, D. L.; Gregor, F.; Arnautov, S.; Kunkel, R.; Bhatotia, P.; Fetzer, C.: secureTF: A Secure TensorFlow Framework. In: Proceedings of the 21st International Middleware Conference. Middleware '20, Association for Computing Machinery, New York, NY, USA, pp. 44–59, Dec. 7, 2020, ISBN: 978-1-4503-8153-6.
- [Sc15] Schuster, F.; Costa, M.; Fournet, C.; Gkantsidis, C.; Peinado, M.; Mainar-Ruiz, G.; Russinovich, M.: VC3: Trustworthy Data Analytics in the Cloud Using SGX. In: 2015 IEEE Symposium on Security and Privacy. 2015 IEEE Symposium on Security and Privacy. Pp. 38–54, May 2015.
- [Su21] Sun, Y.; Wang, S.; Li, H.; Li, F.: Building Enclave-Native Storage Engines for Practical Encrypted Databases. Proceedings of the VLDB Endowment 14/6, pp. 1019–1032, Apr. 12, 2021, ISSN: 2150-8097.
- [SZ15] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, Apr. 10, 2015, arXiv: 1409.1556 [cs].
- [TB19] Tramèr, F.; Boneh, D.: Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. In: 7th International Conference on Learning Representations, ICLR. New Orleans, LA, USA, May 6–9, 2019, arXiv: 1806.03287 [cs, stat].
- [Th22] The Gramine Project: Gramine, 2022, URL: <https://gramineproject.io/>, visited on: 11/24/2022.
- [TPV17] Tsai, C.-C.; Porter, D. E.; Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In: 2017 USENIX Annual Technical Conference (USENIX ATC 17). Pp. 645–658, 2017, ISBN: 978-1-931971-38-6, URL: <https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai>, visited on: 11/24/2022.
- [VGG19] Vinayagamurthy, D.; Gribov, A.; Gorbunov, S.: StealthDB: A Scalable Encrypted Database with Full SQL Query Support. Proceedings on Privacy Enhancing Technologies 2019/3, pp. 370–388, July 1, 2019, ISSN: 2299-0984.
- [Xu21] Xu, T.; Zhu, K.; Andrzejak, A.; Zhang, L.: Distributed Learning in Trusted Execution Environment: A Case Study of Federated Learning in SGX. In: 2021 7th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC). Pp. 450–454, Nov. 2021.



# Inter-Query Parallelism on Heterogeneous Multi-Core CPUs

## Experience Report

Felix Schuhknecht<sup>1</sup>, Tamjidul Islam<sup>2</sup>

### Abstract:

Traditional multi-core CPU architectures integrate a set of homogeneous cores, where all cores are of exactly the same type. With the release of Intel's 12th generation Core x86\_64 processors, this setup has finally changed in the realm of commodity hardware: Apart from so-called performance cores, which provide a high clock frequency, hyper-threading, and large caches, the architecture also integrates so-called efficient cores, which are less performant but rather energy efficient. Obviously, such a performance-heterogeneous architecture complicates task-to-resource scheduling and should be actively considered by the application that schedules the tasks. In this experience report, we discuss our first steps with this new architecture in the context of parallel query processing. We focus on inter-query-parallelism, where whole transactions/queries are the unit of schedule, and investigate which type of core fits to which type of workload best. To do so, we first perform a set of micro-benchmarks on the cores to analyze their different performance characteristics. Based on that, we propose two scheduling strategies that actively schedule tasks to different core types, depending on their characteristics. Our initial findings suggest that the awareness of heterogeneous CPU architectures must indeed be actively incorporated by the task scheduler within a DBMS to efficiently utilize this new type of hardware.

**Keywords:** Query Processing; Parallelism; CPU Architectures; Heterogeneous Cores

## 1 Introduction

For many years, multi-core architectures used to be homogeneous in that they consist of a set of compute cores that all have exactly the same characteristics. This simplified the scheduling problem, as the choice to schedule a task to a specific core was solely determined by the current load and the locality of work to the core.

This situation drastically changed with the advent of heterogeneous multi-core architectures, a development largely driven by the so-called dark silicon effect [Hal11]. This effect describes that thermal and energetic limitations force the CPUs developers to tune down compute units, if they want to increase the overall core count. Initially, heterogeneous designs added

---

<sup>1</sup> Johannes Gutenberg University Mainz, Institute of Computer Science, Staudingerweg 9, 55128 Mainz, Germany  
schuhknecht@uni-mainz.de

<sup>2</sup> Johannes Gutenberg University Mainz, Institute of Computer Science, Staudingerweg 9, 55128 Mainz, Germany  
tislam@students.uni-mainz.de

specialized cores with vastly different feature sets. For instance, the Sparc M7 [Co23] combines general-purpose x86\_64 cores with ASIC data analytics accelerators that support only scans, selections, and semi-joins. Such feature-heterogeneous designs require a careful rethinking of the scheduling mechanism [Du19] as it has to factor in which tasks can actually be carried out by which core. Last year, even mainstream CPUs started to implement heterogeneous multi-core architectures in form of the AlderLake architecture, which resembles Intel’s 12th generation Core consumer processor. Therein, while all cores are general-purpose x86\_64 chips, the architecture separates them into performance cores and efficiency cores. While the faster performance cores are meant to take over highly demanding compute tasks, the efficiency cores should save energy when executing less demanding or less performance-critical tasks. In this sense, they implement a performance-heterogeneity instead of a feature-heterogeneity, on which we will focus in the following report.

As shown in Figure 1, the two core types of an AlderLake i9-12900K highly differ in clock-speed range, cache hierarchy/cache sizes, and whether hyper-threading is available or not, potentially resulting in very different performance characteristics.

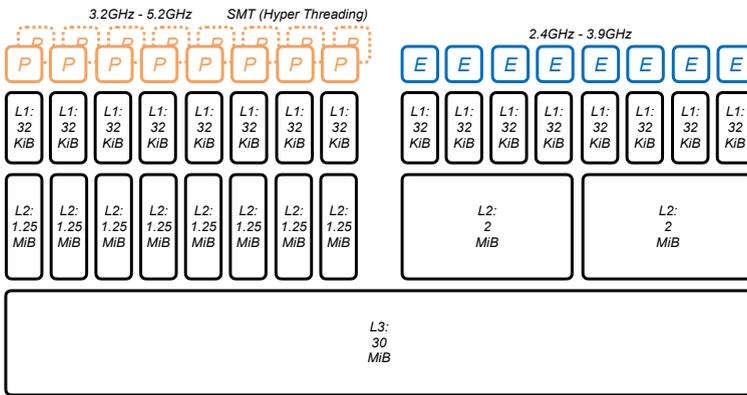


Fig. 1: A modern heterogeneous multi-core architecture (Intel AlderLake i9-12900K).

Therefore, we advocate that even though all cores implement the same instruction set, a scheduler should still be aware of the performance difference of both types of cores and assign tasks actively to the most fitting core type, as we visualize it in Figure 2. Note that operating systems such as Windows 11 and Linux (since kernel 5.18) recently became aware of the heterogeneous design and try to schedule tasks based on classification (a concept marketed as Thread Director [SP22]). In recent years, alternative general-purpose OS-level task scheduling mechanisms tailored to heterogeneous CPU architecture have been proposed [Fa21, SNN22, Ni22, THW02, CJ09, Cr12, AA17]. Typically, they are designed to optimize arbitrary heterogeneous architectures outside the database context and focus on limiting energy consumption, the amount of thread migration, and the runtime of a batch of tasks. We also want to point out that Intel’s new architecture is not the first performance-

heterogeneous CPU: The ARM big.LITTLE, released several years ago, followed a similar principle. In this regard, [Mü14] analyzed how to schedule DBMS-pipelines efficiently to the different core types to optimize for energy-efficiency. With the concept now arriving on commodity x86\_64 CPUs, we see the topic worth to be revisited.

The question remains whether multi-threaded applications such as DBMSs should additionally actively schedule threads on specific performance-heterogeneous cores based on their available domain knowledge on these new processors. As queries have vastly different characteristics, e.g., being compute-bound, bandwidth-bound, short/long-running, or read/write-heavy, it is likely that certain query types will utilize a certain type of core best.

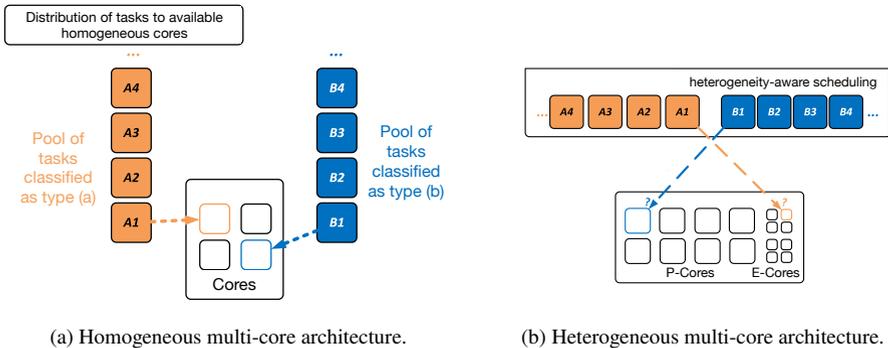


Fig. 2: Task scheduling on uniform architectures (2a) vs heterogeneous architectures (2b).

## 1.1 Contributions and Structure

Therefore, in the following, we would like to investigate the following questions, which also mark the contributions of this experience report:

- (1) Is there a sufficient performance difference between performance cores and efficiency cores that would justify a manual core-type-aware scheduling within a DBMS?
- (2) Which type of tasks perform well on which type of core? For which type of tasks is the core choice irrelevant?
- (3) Do core-type-aware scheduling strategies (push-based vs pull-based) perform better than a completely unaware strategy?

The experience report is structured as follows: In Section 2, we first perform an initial benchmarking of the AlderLake architecture, in which we identify how different workloads map to the available cores. In Section 3, we present and evaluate two heterogeneity-aware scheduling strategies and compare them against an unaware strategy. In Section 4, we outline possible next steps in this topic and conclude with our early findings.

## 2 Benchmarking the Architecture

We start by performing a set of micro benchmarks on our performance-heterogeneous AlderLake CPU to identify which workload fits to which type of core. For all upcoming experiments, we use an Intel i9-12900K with 8 performance cores of up to 5.2GHz (with SMT aka hyper-threading) and 8 efficiency cores of up to 3.9GHz (without SMT). This results in a total of 16 physical and 24 logical cores. The machine is equipped with 128GB of DDR4-3200 RAM. As operating system, we use Arch Linux running kernel 5.17.5. Note that in this kernel version, the scheduler was not yet aware of the heterogeneous architecture – for our experiments, this does not matter as we manually perform all thread assignment in our code. A comparison with a heterogeneity-aware scheduler on a newer kernel is left for future work.

Figure 3 shows the results for executing four different workloads on each available logical core individually. Note that logical cores 0-15 resemble the (logical) performance cores, whereas cores 16-23 are the efficiency cores. No parallelism is happening here as only one core is active during each measurement. To get an intuition for the architecture, we perform the following set of micro-benchmarks on 300M integers in total: In Figure 3a, we schedule the sorting of an array with integers using `std::sort`. In Figure 3b, we copy randomly selected integers from one array into a second array. In Figure 3c, we copy the entire array of sequentially into another array using `memcpy`. In Figure 3d, we read the entire array sequentially to compute the sum of all integers. As we can see, the tasks have a different runtime on the individual logical cores, where some tasks are highly affected by the type of core while other tasks hardly show a performance difference at all. The sorting task, being compute heavy and containing random access clearly benefits from being executed on a performance core (core 0-15). Also, random copies perform better on performance cores, although the difference being less prominent. When looking at the sequential tasks, we interestingly hardly notice any difference between the core types anymore, as these tasks are rather bandwidth bound than compute bound.

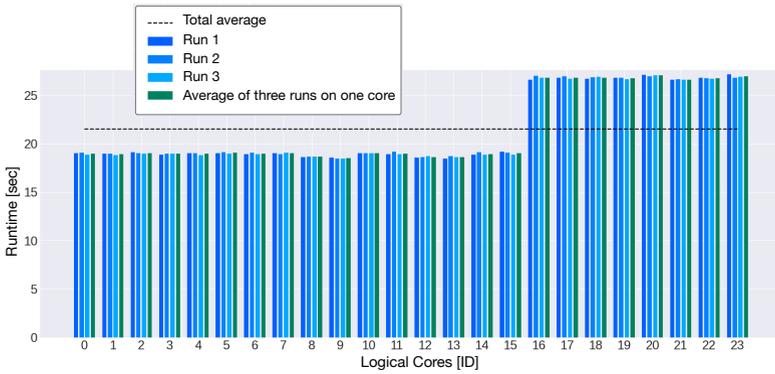
Overall, this gives us a first recommendation on how to utilize the cores: Compute-bound tasks or tasks that contain random access should preferably go to performance cores, while sequential tasks, that are mostly bandwidth-bound, could also be scheduled on efficiency cores.

## 3 Heterogeneity-aware Scheduling

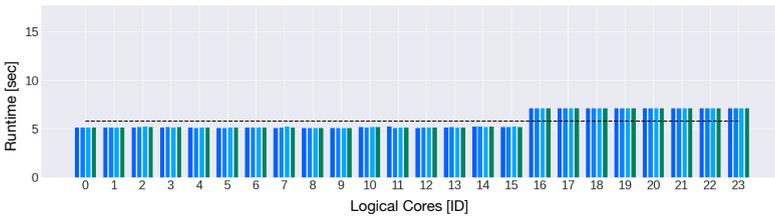
With the gained knowledge, in the following, we build and evaluate a simple scheduling mechanism for parallel query processing using two different scheduling strategies.

### 3.1 Setup and Task Types

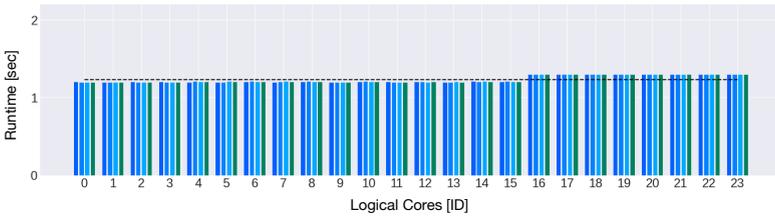
We set up a table of  $n$  integers columns in row-layout represented by a two-dimensional vector and support two types of tasks on this table: **(a) Updating transactions** that modify



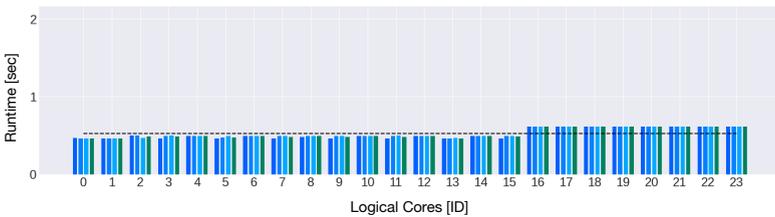
(a) Sorting (compute-heavy, random reads, random writes)



(b) Copying randomly selected integers (random reads, random writes)



(c) Copying integers sequentially (sequential reads, sequential writes)



(d) Scanning & Aggregating (sequential reads)

Fig. 3: Executing different types of workloads of each available logical core.

a certain number of rows of the table. These resemble a task from a traditional transactional workload. **(b) Read-only queries** that select a specific value of a specific column and count how often this value has been seen. These could represent a typical query from an analytical workload. When creating a test workload, we specify the ratio of tasks of type (a) in relation to tasks of type (b) and fill two pools of pending tasks of each type accordingly. Our scheduler then executes these tasks using a specific scheduling strategy.

### 3.2 Scheduling Strategies

We compare three different strategies in the following. The first strategy is push-based and simply ignores the heterogeneous architecture, as it uniformly distributed tasks of both types to available cores. It will serve as our baseline. Figure 4 visualizes the setup for a batch of 100 tasks. Each core maintains a pool for update transactions and read-only queries.

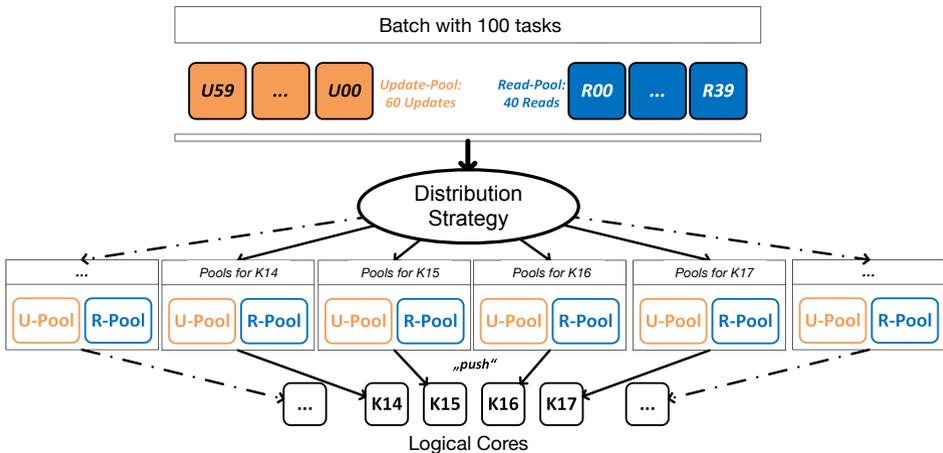


Fig. 4: Architecture of the push-based scheduler. As distribution strategy, we test both a non-aware strategy, which uniformly distributes tasks to all cores and an aware strategy, which tries to schedule update transactions to performance cores and read-only queries to efficiency cores.

The second strategy is also push-based and resembles the architecture of Figure 4, but does not uniformly distribute tasks across cores. Instead, it is aware of the heterogeneity and tries to assign update transactions to performance-cores and read-only queries to efficiency cores. Only if no core of the respective type is currently available, the task is scheduled on the other type of core. The availability of cores is recorded in status flags that are accessed by the scheduler and updated by the threads running on the cores.

The third strategy is also heterogeneity-aware, but follows a pull-based approach as visualized in Figure 5. Here, the performance cores preferably pull from the pool containing update transactions, while the efficiency cores pull from the pool containing read-only queries.

Only if no work is left in a pool, the other pool is considered. A mutex protects each pool to avoid races during the pulling of tasks.

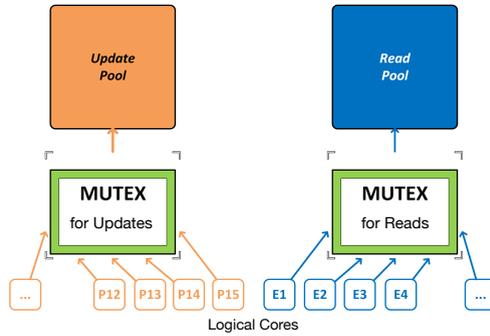


Fig. 5: Pull-based scheduler that is aware of the heterogeneous architecture.

Let us now see how the three scheduling strategies perform in comparison. We fire a batch of 180 tasks and vary the mixture between update transactions and read-only queries in steps of 25%. We use a table with 60M rows and 150 columns. Every update transaction updates 12M randomly selected rows. We observe that the strategy indeed has a significant impact on the runtime. The more update transactions we have in our batch, the more the strategy matters and the heterogeneity-aware strategies win. This makes sense, as update transactions must be actively scheduled to performance cores to yield the best runtime.

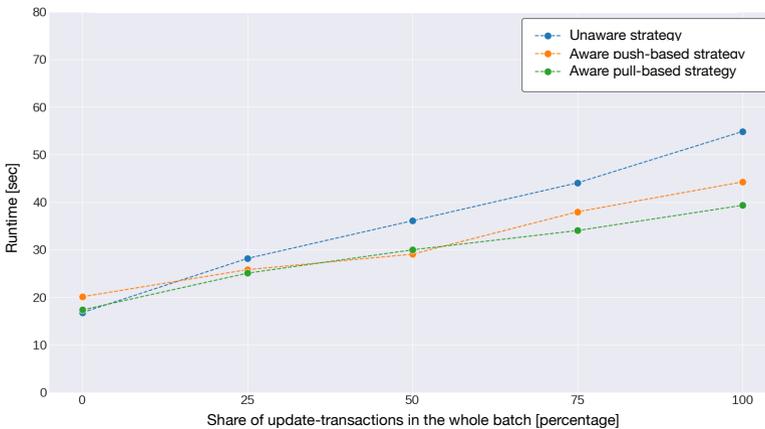


Fig. 6: Comparison of scheduling strategies under a varying mixture of update transactions and read-only queries.

To get a deeper insight in the behavior of the task scheduling, in Figure 7 we additionally plot heatmaps for the cases of 25% update transactions and 75% read-only queries (Figure 7a)



types of cores. The reason for this is that due to the low number of update transactions, the performance cores are soon unused and can be used to answer read-only queries as well. This behavior is different for the pull-strategy, where read-only queries are pulled only by efficiency cores in the run. In Figure 7b, we see a similar picture, however, more compute-intensive update transactions must be handled. These are indeed primarily scheduled to the performance cores by both strategies. When analyzing the whole batch, we also observe that using heterogeneity-aware strategies homogenizes the latency of individual tasks in a batch. For the naive strategy, the minimum and maximum latency differs highly, while this difference is much smaller for the aware strategies, showing that the hardware resources are efficiently utilized.

## 4 Outlook and Conclusion

In this experience report, we presented our initial steps in understanding the impact of a performance-heterogeneous CPU design on parallel query processing. To simplify the analysis, we focused on inter-query parallelism, i.e., we scheduled whole transactions/queries to individual cores and evaluated two strategies that try to cleverly assign fitting tasks to a specific core type. We tested two simple heterogeneity-aware scheduling strategies and showed that even on this coarse-grained level, a measurable performance boost can be observed over an unaware strategy.

Of course, this report marks only the very first step towards query parallelism on this type of hardware. As modern DBMS schedulers typically break down a transactions/query into more fine-granular compiled pipelines and schedule these individually [Le14, NF20, Ne21], a next step is to extend such a pipeline scheduler with heterogeneity-awareness. This involves on-the-fly classification of (arbitrary) pipelines, finding a suitable mapping between pipelines and core types, handling dependencies between pipelines, and ensuring a constant utilizations of all cores. Also, an important baseline for any fine-grained approach will be the new OS-level heterogeneity-aware scheduler. Here, we see a major advantage of a manual approach by being able to include domain knowledge, such as detailed transaction/query behavior, into the scheduling process. However, such a comparison is left for future work.

## Bibliography

- [AA17] AlEbrahim, Shaikhah; Ahmad, Imtiaz: Task scheduling for heterogeneous computing systems. *J. Supercomput.*, 73(6):2313–2338, 2017.
- [CJ09] Chen, Jian; John, Lizy Kurian: Efficient program scheduling for heterogeneous multi-core processors. In: *Proceedings of the 46th Design Automation Conference, DAC 2009, San Francisco, CA, USA, July 26-31, 2009*. ACM, pp. 927–930, 2009.
- [Co23] Corporation, Oracle: Oracle’s SPARC T7 and SPARC M7 Server Architecture: <https://www.oracle.com/assets/sparc-t7-m7-server-architecture-2702877.pdf>. 2023.

- [Cr12] Craeynest, Kenzo Van; Jaleel, Aamer; Eeckhout, Lieven; Narváez, Paolo; Emer, Joel S.: Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In: 39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA. IEEE Computer Society, pp. 213–224, 2012.
- [Du19] Dursun, Kayhan; Binnig, Carsten; Çetintemel, Ugur; Swart, Garret; Gong, Weiwei: A Morsel-Driven Query Execution Engine for Heterogeneous Multi-Cores. *Proc. VLDB Endow.*, 12(12):2218–2229, 2019.
- [Fa21] Fan, Zhichao; Hu, Wei; Guo, Hong; Liu, Jing; Gan, Yu: An Efficient Scheduling Algorithm for Interdependent Tasks in Heterogeneous Multi-core Systems. In: 2021 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2021, Melbourne, Australia, October 17-20, 2021. IEEE, pp. 2354–2359, 2021.
- [Ha11] Hardavellas, Nikos; Ferdman, Michael; Falsafi, Babak; Ailamaki, Anastasia: Toward Dark Silicon in Servers. *IEEE Micro*, 31(4):6–15, 2011.
- [Le14] Leis, Viktor; Boncz, Peter A.; Kemper, Alfons; Neumann, Thomas: Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In (Dyreson, Curtis E.; Li, Feifei; Özsu, M. Tamer, eds): International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014. ACM, pp. 743–754, 2014.
- [Mü14] Mühlbauer, Tobias; Rödiger, Wolf; Seilbeck, Robert; Kemper, Alfons; Neumann, Thomas: Heterogeneity-conscious parallel query execution: getting a better mileage while driving faster! In (Kemper, Alfons; Pandis, Ippokratis, eds): Tenth International Workshop on Data Management on New Hardware, DaMoN 2014, Snowbird, UT, USA, June 23, 2014. ACM, pp. 2:1–2:10, 2014.
- [Ne21] Neumann, Thomas: Evolution of a Compiling Query Engine. *Proc. VLDB Endow.*, 14(12):3207–3210, 2021.
- [NF20] Neumann, Thomas; Freitag, Michael J.: Umbra: A Disk-Based System with In-Memory Performance. In: 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings. [www.cidrdb.org](http://www.cidrdb.org), 2020.
- [Ni22] Nishikawa, Hiroki; Shimada, Kana; Taniguchi, Ittetsu; Tomiyama, Hiroyuki: Simultaneous Scheduling and Core-Type Optimization for Moldable Fork-Join Tasks on Heterogeneous Multicores. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 105-A(3):540–548, 2022.
- [SNN22] Salami, Bagher; Noori, Hamid; Naghibzadeh, Mahmoud: Online energy-efficient fair scheduling for heterogeneous multi-cores considering shared resource contention. *J. Supercomput.*, 78(6):7729–7748, 2022.
- [SP22] Saez, Juan Carlos; Prieto-Matías, Manuel: Evaluation of the Intel thread director technology on an Alder Lake processor. In (Serafini, Marco; Xu, Harry, eds): APSys '22: 13th ACM SIGOPS Asia-Pacific Workshop on Systems, Virtual Event, Singapore, August 23 - 24, 2022. ACM, pp. 61–67, 2022.
- [THW02] Topcuoglu, Haluk; Hariri, Salim; Wu, Min-You: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distributed Syst.*, 13(3):260–274, 2002.

# An FPGA Avro Parser Generator for Accelerated Data Stream Processing

Tobias Hahn,<sup>1</sup> Daniel Schüll,<sup>2</sup> Stefan Wildermann,<sup>3</sup> Jürgen Teich<sup>4</sup>

**Abstract:** Big Data applications frequently involve processing data streams encoded in semi-structured data formats such as JSON, Protobuf, or Avro. A major challenge in accelerating data stream processing on FPGAs is that the parsing of such data formats is usually highly complex. This is especially true for JSON parsing on FPGAs, which lies in the focus of related work. The parsing of the binary Avro format, on the other hand, is perfectly suited for being processed on FPGAs and can thus serve as an enabler for data stream processing on FPGAs. In this realm, we present a methodology for parsing, projection, and selection of Avro objects, which enforces an output format suitable for further processing on the FPGA. Moreover, we provide a generator to automatically create accelerators based on this methodology. The obtained accelerators can achieve significant speedups compared to CPU-based parsers, and at the same time require only very few FPGA resources.

**Keywords:** Avro, parsing, FPGA, semi-structured data, accelerator

**Acknowledgement:** This work has been supported by the German Science Foundation (Deutsche Forschungsgemeinschaft, DFG) as part of the Priority Programme SPP 2037.

## 1 Introduction

Many Big Data applications in areas such as the Internet of Things and Industry 4.0 are not only confronted with large volumes of data generated at a high frequency but also place high demands on the latency for analyzing this data. In this area, stream processing is becoming increasingly important, which means that data is continuously processed and analyzed as soon as it is generated or received. To meet the growing demands of stream processing applications in terms of high throughput and low latency, FPGA accelerators have been proposed as a solution in the past [MTA09; TM11]. Since stream applications typically run for long periods, the relatively long synthesis times required to generate application-specific accelerators are tolerable as they enable perfectly tailored and thus very resource- and energy-efficient data processing.

For being able to build such FPGA accelerators, different approaches to compile queries to FPGA primitives have been presented in the past [MTA09; MTA10; Sa12]. However,

---

<sup>1</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany, tobias.hahn@fau.de

<sup>2</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany, daniel.schuell@fau.de

<sup>3</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany, stefan.wildermann@fau.de

<sup>4</sup> Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany, juergen.teich@fau.de

these approaches ignore the fact that the data arriving at the FPGA is usually formatted in ways that are difficult to process directly on the FPGA or even by machines in general. As a matter of fact, parsing such data may take most of the time when being processed on a CPU. For example, in case of JSON, it has been shown that parsing may account for around 90% of CPU time for some stream processing applications [Li17].

Offloading parsing to an FPGA would offer two major advantages. First, would relieve the CPU from this time-consuming task so that it can be better utilized by other tasks or workloads. Second, this would be particularly advantageous when FPGAs can directly access the data stream, e.g., when attaching them to a network interface or in the form of FPGA-based smart NICs, as additional data movements could be avoided. However, this requires concepts to parse the serialized data format as a byte stream.

In this realm, two approaches to parsing JSON data on FPGAs have been presented recently [Da22; Pe21]. However, the presented approaches only consider the acceleration of the parsing process, thus supporting only traditional software-based data processing. Consequently, they parse the received data into data structures that are tailored to be further processed on a CPU. In contrast, the approach presented in this paper aims at enabling complete data processing on FPGAs.

We specifically target parsing, selection, and projection of Avro objects which are widely used in Apache-based computing infrastructures. The Avro format [Ap21] has a much higher information density than semi-structured formats such as JSON. It is better tailored to FPGA-based processing than to CPU-based processing, as techniques to increase the performance of modern CPUs, like branch prediction, multi-level caches, and SIMD instructions, are not of any benefit when parsing Avro data. In this paper, we present general techniques for parsing data streams consisting of Avro objects on FPGAs. In addition, we introduce a methodology to automatically generate hardware accelerators for parsing, selection, and projection on FPGAs based on user-defined Avro schemas and queries. We present a system architecture where accelerators generated with the methodology can be loaded to parse data streams of Avro objects at line rate, for example, arriving at a network interface. Thanks to a fixed data layout, the outgoing data stream can even be forwarded to other FPGA accelerators and used to process further application steps. Moreover, due to low resource consumption, sufficient resources are often remaining available on the FPGA to build further accelerators for subsequent processing of the parsed, filtered, and projected data stream.

The paper is organized as follows: First, in Sect. 1.1, we will give a brief overview of common semi-structured data formats and justify our choice of Avro. In Sect. 2 we will introduce our parser generator and describe how an accelerator can be created given a schema and a path expression. In Sect. 3, the obtained accelerators are evaluated using the Yahoo [Ch16] and RiotBench [SCS17] benchmarks. Finally, the paper closes in Sect. 4 with a conclusion and an outlook for future work.

## 1.1 Data Formats for Stream Processing

Despite the huge overheads that parsing semi-structured data formats imply, there are many good reasons why these formats are nonetheless used universally today. These can be broken down to being human readable, having a relatively small data footprint, the possibility to enforce a fixed schema, and the possibility of extending the format while maintaining forward and backward compatibility (*schema evolution*).

Semi-structured data formats can be roughly divided into *storage formats* and *exchange formats*. Storage formats arrange objects in such a way that they can be quickly searched for individual attributes, e.g., by using column-oriented formats. Examples are Apache Parquet [Ap22b], Apache ORC [Ap13], and Google Dremel [Me10]. The exchange formats on the other hand are easy to be serialized which makes it possible to write and process objects continuously as a data stream. Examples are JSON [Br17], Apache Avro [Ap21], and Google Protocol Buffers [Go22]. In the following overview, we focus on exchange formats, as we aim to accelerate data stream applications.

**JSON, CBOR & Protobuf** Tab. 1 gives an overview of the most common exchange formats. For each format, it rates the (a) readability by human and by machine and (b) the data footprint. Here, also the footprint for encoding the same record is displayed. List. 1 shows this record formatted in JSON. Finally, the table rates (c) schema evolution and (d) whether the format uses a schema. The most commonly used exchange format is the text-formatted JSON [Br17], which particularly stands out due to its high human readability. However, this in turn has a severe negative effect on its data footprint. In addition, JSON can also be used to achieve a good backward and forward compatibility by simply adding new attribute fields in newer versions. CBOR [BH20] is also a schema-less format where fields are binary encoded, resulting in a smaller data footprint, while still allowing attributes to be added or omitted as desired. Protobuf, on the other hand, relies on binary encoding and a schema that can be extended without corrupting older parser versions. This is achieved by assigning an index to all attributes in the schema so that the old parser versions can simply ignore indexes they do not recognize.

Tab. 1: Overview of the strengths, weaknesses, and general properties of data formats discussed.

|          | readability |         | footprint (car ex. size) | schema evolution | schema used |
|----------|-------------|---------|--------------------------|------------------|-------------|
|          | human       | machine |                          |                  |             |
| JSON     | ++          | --      | -- (96B)                 | ++               | no          |
| CBOR     | --          | +       | - (50B)                  | ++               | no          |
| Protobuf | -           | ++      | + (18B)                  | ++               | yes         |
| Avro     | -           | ++      | ++ (12B)                 | +                | yes         |

```

{
  "id": 42,
  "name": "Golf",
  "engine": {
    "serialNr": 1234,
    "horsepower": 85.5
  }
}
{
  "name": "car",
  "type": "record",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "engine", "type": {
      "name": "engine",
      "type": "record",
      "fields": [
        {"name": "serialNr", "type": "int"},
        {"name": "horsepower", "type": "float"}
      ]
    }
  ]
}

```

List. 1: Motivational JSON record.

List. 2: Avro schema for a car object.

**Avro** Avro [Ap21] is a binary schema-based data format. Unlike Protobuf, Avro does not use indexes to identify fields and consequently requires even fewer bytes for encoding. The type and order of the fields are defined solely by the schema used. Accordingly, the Avro object encoding itself does not support schema evolution, as decoding always requires the corresponding schema. Instead, Avro is usually used in combination with wrapper formats, as presented in Sect. 1.1, to solve schema evolution at a higher protocol layer.

In the following, the Avro specification [Ap21] will be presented in more detail. Avro offers a range of elementary and complex data types. As elementary data types, booleans, signed integers (32-bit), signed longs (64-bit), floats (32-bit), doubles (64-bit), strings, and byte sequences of fixed and variable length are available. Avro includes records, enums, arrays, maps of key-value pairs, and union types as complex types. The Avro schema to be used is specified in JSON. List. 2 shows the Avro schema that complies with the JSON record from List. 1.

In terms of parsing speed, Avro outperforms all other formats. JSON is the most time-consuming to parse due to its text-based format. Unlike Protobuf and Avro, both JSON and CBOR cannot be parsed using finite state machines due to the arbitrarily deep nesting of records, making parsing again more complex. Since no indexes or attribute names are required for reading, parsing Avro has the advantage over Protobuf that only the sequence of fields defined in the schema has to be processed. However, Avro parsing does neither require complicated control flow nor flexible memory accesses and thus is not the field for which modern general-purpose processors with their sophisticated branch prediction and multi-level cache hierarchy have been optimized for. Nonetheless, a required simple finite state machine can be easily mapped to FPGAs with very low resource requirements. In this paper, we show in this paper how the generation of such logic circuits can be performed completely automatically solely based on the specified schema and a query defining projection and selection in a path expression.

**Avro Container- & Wire-Format** The Avro format can be used both for storing data on a hard drive and for transferring data over a network. In most cases, two wrapper formats are used, which are tailored to the respective case. For the storage of Avro objects, there is the Avro Object Container Format, which stores the used schema directly beside the data. A particular advantage of this format is the partitioning of the objects into blocks, which enables an efficient separation for parallel processing.

The wire format, on the other hand, is optimized for the transmission of data over a network. The Avro object stream is organized in a series of buffers. Each buffer begins with a four-byte length field that specifies the buffer length in bytes. The respective number of subsequent bytes contain Avro objects. A message may contain multiple buffers. The end of a message is indicated by a buffer of length zero, i.e., only a length field with value 0, and no buffer data is transmitted. In contrast to the container format, the schema is not transmitted here. Instead, the receiver must already be familiar with it, e.g., Apache Kafka uses a registry to resolve schemas of incoming data.

## 1.2 Related Work

Extensive literature already exists for processing XML data on FPGAs [EI10; Mi09; TWN12; WA11]. XML parsing on FPGAs has always been accompanied by path expressions for projections to reduce the amount and complexity of the outstream data. As Koch et al. [KSS08] have shown, projecting XML data can be solved most efficiently by transforming the problem into a string matching problem. However, this makes it difficult to transfer findings from XML parsing to parsers for binary encoded formats like Avro.

Recently, research attention has shifted to the JSON format which is predominant today. However, research on JSON parsing on FPGAs is still scarce, and furthermore, there is no solution for integrating FPGA parsers with existing accelerators for subsequent query processing on the FPGA. Peltenburg et al. [Pe21] propose to speed up JSON parsing by converting the input data to the columnar in-memory format Apache Arrow [Ap22a]. The FPGA performs the parsing of the JSON data, converts the data to the Arrow format, and then transmits it to the host memory. The authors implement their parser with a one-to-one relation between fields in the schema and parser blocks, which hence expects the same schema at all times. However, there is no fixed schema for JSON, so valid data in terms of the JSON specification can cause the parser to enter an invalid state. PipeJSON [Da22] follows a more flexible approach where the JSON data is converted into a tape structure, similar to CPU-based parsers. This tape structure consists of a variable-length array of 64-bit values, which contains the decoded values as well as offsets for navigating through the array structure (e.g., to the end of a record). While such flexible data structures can be processed efficiently on the CPU, they are unsuitable for further processing on the FPGA. Hahn et al. [Ha22; HWT22] present acceleration techniques for raw filtering of JSON data on FPGAs. Raw filtering is a selection technique on the raw byte stream of serialized JSON data. As such, it does not require to parse JSON records completely, but only to identify

specific patterns according to the filter expression in the byte stream. However, since only irrelevant JSON records are filtered out and the data is not completely parsed, it does not directly enable any further processing of a given data stream on the FPGA.

## 2 Proposed Parsing Architecture

In this section, we present a technique for automatically generating hardware parser accelerators for a given schema and query (specified by JSONPath [Gö07]). The schema defines all elements that appear in each Avro object. JSONPath defines the selection criteria and attributes to be projected per Avro object. The parser generator goes through three phases. In the first phase, an object parser is created that can parse incoming Avro objects (see Fig. 1 left). In the second phase, this object parser is then extended to a Parse, Project & Select (PPS) module by augmenting logic for projection and selection (see Fig. 1 right). During the third phase, this PPS module is then wrapped into an accelerator which can later be deployed on the FPGA.

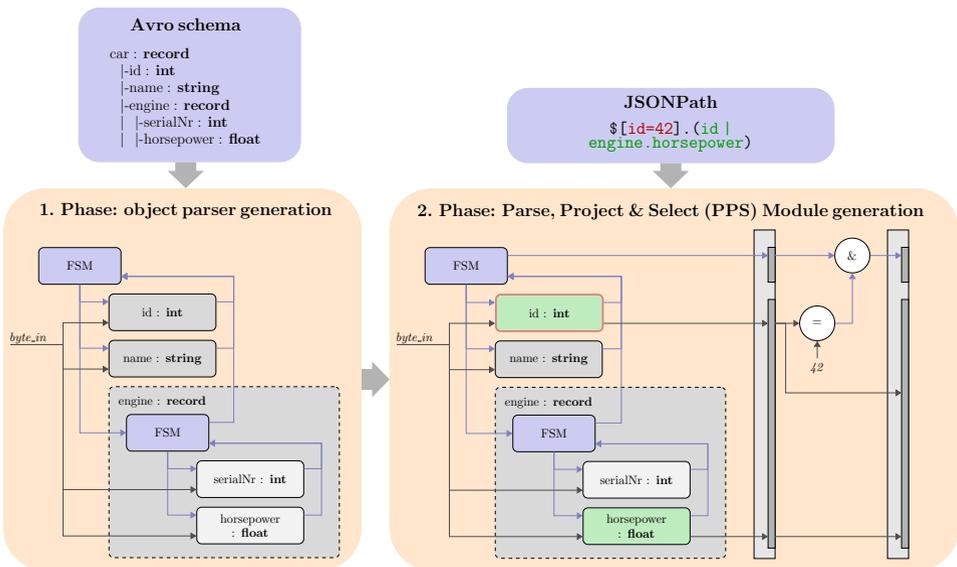


Fig. 1: Overview of the first and second phase of the parser generation process.

In the *first phase*, the schema is interpreted and a corresponding object parser is generated. For this purpose, the schema is traversed recursively. A parser block module is instantiated for each field of the schema. A finite state machine coordinates which field in the schema, respectively which parser block module is responsible for parsing the incoming byte at each clock cycle.

*Example:* Fig. 1 (left) schematically illustrates the parser structure generated for the schema given in List. 2. It contains blocks for `id`, `name`, and `engine`, as well as an FSM that coordinates the sequential parsing of the respective blocks. The latter is a complex (recursive) type, which again consists of internal parser blocks as well as an FSM to hierarchically coordinate the parsing procedure. Sect. 2.1 explains the details of parser block generation.

In the *second phase* (see Sect. 2.2), the PPS module is generated based on the object parser from the first phase and the specified JSONPath query, which defines the attributes to be projected and the selection criteria. First, all parser blocks are determined that provide the attributes required for selection or projection and their output signals are connected to a register stage (*stage 1*). The register stage also contains a valid bit which indicates whether the object was read completely and that the data in the register represents a valid Avro object. Subsequently, a further pipeline stage (*stage 2*) is generated containing the logic for evaluating the selection expressions by comparisons on the values stored in the stage 1 register and combining the result with the valid bit of the stage 1 register.

*Example:* Fig. 1 (right) shows an example query with a selection on field `id` and projection of attributes `id` and `horsepower` together with the generated parser accelerator. Only the signals of the parser blocks responsible for parsing the respective attributes get connected to the register stages. The selection logic for `id == 42` is added between the register stages.

In the *third phase* (see Sect. 2.3), an accelerator is generated from the PPS module created in the second phase, which can finally be deployed on the FPGA. One of the challenges of parsing Avro data is that some of the elementary data types are encoded with variable lengths. This makes it very difficult to parallelize generated hardware components to process multiple bytes in one cycle since in order to interpret a byte, it must first be clear which field in the schema it corresponds to. Therefore, we choose to process only one byte per cycle with the introduced object parsers and PPS modules. During the third phase, however, we again introduce parallelism by inserting multiple parallel PPS modules in the accelerator. For this purpose, we exploit the properties of the wire format to split Avro objects among parallel channels, while working with a higher word width.

## 2.1 Phase 1 – Object parser generation

Avro schemas are composed of different elementary and complex types. In the following, we present parser blocks for each Avro type, excluding the null and array type. Parser blocks can likewise be divided into *elementary parser blocks* (`fixed`, `float`, `boolean`, `int`, `enum` & `string`) and *complex parser blocks* (`record`, `map` & `union`). Elementary parser blocks are the basic blocks that interpret the input bytes to parse the desired fields. Complex parser blocks, on the other hand, are composed of one or more parser blocks (both elementary & complex) and do not interpret any input data but merely coordinate when which of the contained blocks is *active* and when its output is *valid*.

All parser blocks follow the same interface as illustrated in Fig. 2. This consists of an input `in_byte`, which contains the current input byte of the Avro object and is directly connected across all parser blocks. The port `in_valid` controls which parser block is active. The activated block accordingly interprets the obtained data on the port `in_byte`. If a parser block finished reading an encoded field, the port `out_valid` is set to one. The signal `out_value` remains still unconnected during the first phase and is subsequently connected to the register stages during the second phase in case it is required for selection or projection. Next, the details for generating parser blocks for covering all supported Avro types.



Fig. 2: Interface of a parser block.

**Fixed parser block (fixed)** The *fixed* parser block reads a fixed amount of  $n$  bytes, which is statically defined by the given schema. The block is controlled based on a counter which is initialized with  $n - 1$  and decremented in each clock cycle the signal `in_valid` is one. Each input byte is written into a shift register of size  $n - 1$  bytes. When the counter reaches zero, the complete field is available and a one is emitted on `out_valid`. The signal `out_value` with  $n$  bytes is then composed of the concatenation of the signal `in_bytes` and the  $(n - 1)$  bytes in the shift register.

**Float/double and Boolean parser block (float and boolean)** In the Avro format, floating point numbers are directly encoded in the IEEE 754 format. Accordingly, a *float* parser block is just a special case of a *fixed* parser block of constant size  $n$ , which is 4 bytes for floats (single precision) and 8 bytes for doubles (double precision). No further binary format conversion is necessary by this parser block. The same applies to the *boolean* parser block, which is always encoded using one byte and therefore implemented as *fixed* parser block with a constant size of 1.

**Int/long and enum type parser block (int and enum)** Avro integers and longs are encoded via the zigzag format, which allows for a small data footprint. The zigzag format is a variable-length quantity code in which both small positive and negative numbers can be encoded to fewer bytes. This is achieved by reserving the eighth bit of each byte to indicate whether there are more bytes to follow, as can be seen in the example integer in Fig. 3. The remaining parts carry payload bits. The task of the *int* parser block is to take the zigzag formatted integer byte by byte and to decode it into the two's complement.

For the sake of clarity, however, let's first consider a zigzag-formatted number  $z$  without byte boundaries or continuation bits. The decoding of this zigzag encoded number  $z$  into

its corresponding two's complement  $b$  of constant size  $n$  depends on whether a positive or negative number is encountered. Accordingly, the first step is to extract the sign of the number to be decoded, which is located in bit 0 of the least significant byte ( $z[0]$ ). Subsequently, the decoding is done via

$$b = \begin{cases} z \gg 1 & \text{if } z[0] = 0 // \text{positive} \\ \sim (z \gg 1) & \text{else //negative} \end{cases}$$

where  $\gg$  is a non-arithmetic right shift (i.e., 0 padding) and  $\sim$  is a bitwise inversion.

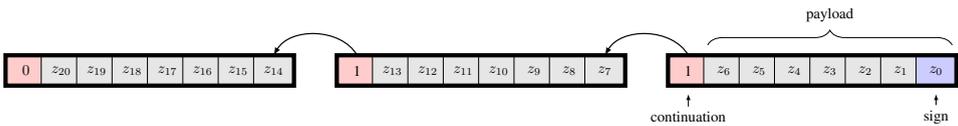


Fig. 3: Example of a 3 byte zigzag integer.

As the decoding is carried out byte by byte, the sign is extracted at the start of the first step  $k = 0$ . If a 1 is observed, the six payload bits  $z_6$  to  $z_1$  of the first byte are inverted. The result whether inverted or not is then written into the lowest 6 bits of the destination register ( $b_0 \dots b_5$ ). Since the obtained number can be smaller than the bit width of  $b$ , all higher bits ( $b_6 \dots b_{n-1}$ ) must be initially set to 1 in case of a negative sign, resulting in a sign extension. Each following byte  $k > 0$  is treated as follows: 1.) Invert the 7 payload bits in case of a negative sign. 2.) Write the result into the next 7 bits of the destination register ( $b_{6+(k-1)*7} \dots b_{6+k*7-1}$ ). The  $b$  register is then connected to *out\_data* and, as soon as a continuation bit is 0, the signal *out\_valid* is set to 1.

The size of  $b$ , respectively of the signal *out\_data* can be configured freely for the int parser block. Although 4 bytes are always used for integers and 8 bytes for longs, integer parser blocks are also used internally for parsing enums and metainformation fields of other types (see string, maps, union parser blocks below), where smaller sizes may be sufficient. Thus the parser blocks can be constructed as small as possible. An enum block is accordingly implemented as an int block with a  $b$  register size of  $n$  equal to  $\log_2(\# \text{ enum elements})$  bits.

**String parser block (string and count\_byte)** The string parser block consists of an int parser block and a count\_byte block which are processed one after the other as shown in the flowchart in Fig. 4. As soon as the integer length field *len* is parsed, the received value is used to initialize the count\_byte block. This block is basically a counter that remains active for *len* subsequent valid bytes, with the current input byte written to a shift register (similar to the fixed parser block). The length of the shift register, and thus the maximum readable string length, is set to 32 bytes by default but can be altered in the second phase by specifying a maximum string length in the JSONPath.

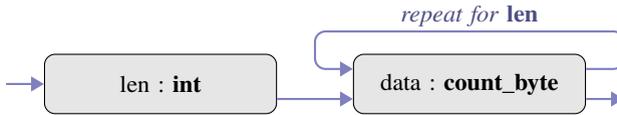


Fig. 4: FSM for the string parser block.

**Record parser block (record)** A record type contains multiple fields each being of a specific Avro type. The record parser block, therefore, contains one parser block for each field. Once *in\_valid* is set to one, a controller consecutively activates the contained parser blocks for evaluating the input bytes, as shown in Fig. 5. The signal *out\_valid* is set to one on completion of the last block.



Fig. 5: FSM for the record&lt;type\_0, type\_1, . . . , type\_n&gt; parser block.

**Map parser blocks (map, key\_value and string\_matcher)** The map type is encoded by an int field *obj\_cnt*, followed by as many key-value pairs as specified in the int field. Once the *obj\_cnt* field is parsed, it is used to initiate the loop counter to control the *key\_value* parse block, as illustrated in Fig. 6. The *key\_value* parse block sequentially parses first a key, which is encoded as a string, and then the respective value, which type is defined in the schema. As long as the loop counter is greater than 0, the *key\_value* parser block is kept active, which repeats its internal parsing. The loop counter is decremented each time the signal *out\_valid* of the *key\_value* parser block is one, i.e., after each parsed key-value pair. The *key\_value* parser block is a special case of a record parser block with only two fields, where the first field is a *string\_matcher* block. This is a special block that can test whether the parsed string matches a given string from a dictionary. The rationale for this is that a query may use values from a subset of keys for projection or selection. This means that only key-value pairs with keys from this set have to be registered for the next pipeline stage. Therefore, the second phase of parser generation will populate the string matcher dictionary with each key string in this subset and augment the corresponding matching logic. The string matcher will generate one signal per key in this set to indicate when the respective key-value pair is currently parsed (see Sect. 2.2 for more details). According to the Avro specification, the specified length of the map can also be negative. In this case, the absolute value of the length does not indicate the number of objects, but the length of the payload data in bytes. However, this behavior is not supported in our parser block but could be added in the future by modifying the loop counter.

**Union parser block (union)** The Avro union type is a complex type. It is possible to specify a list of several different types in the schema for a union field. The data contained in that field of an Avro object can then be of one of these types. This is achieved in Avro

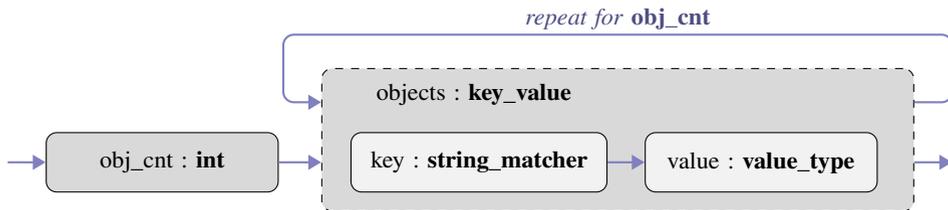


Fig. 6: FSM for the map<key\_type, value\_type> parser block.

by encoding an int index before the payload to indicate which type should be used for encoding the field. The index can then be used to select the desired type from the given list of types during interpretation. Accordingly, the union parser block is composed of an int block followed by all types specified in the schema, respectively their associated parser blocks, as seen in Fig. 7. Once the parsing of the *union\_index* int block is complete, only the input of the respective *type\_(union\_idx)* block is activated. The signals *out\_valid* of the parallel blocks can simply be combined via an or-reduction, as only one of the blocks can become active anyway.

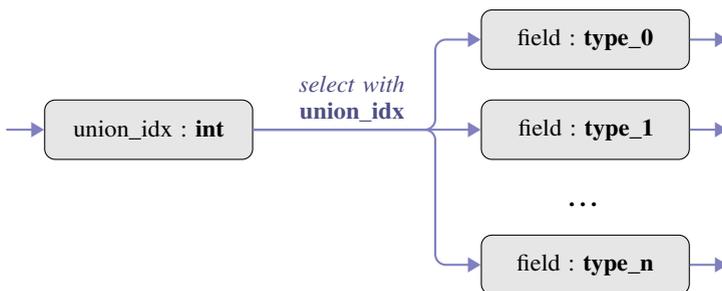


Fig. 7: FSM for the union<type\_0, type\_1, ..., type\_n> parser block.

**Object parser generation** The parser generation consists of allocating parser blocks according to the given schema and generating the control logic of the FSMs (one for each complex parser block) to coordinate the parsing process. Each Avro schema can be represented by a *object parser tree*  $G_O\{V_O, E_O\}$  with vertices  $V_O$  and edges  $E_O$ . The leaves are elementary types. Complex types have multiple children. The child nodes are ordered in the order they appear in the given schema. In the object parser generation phase, the respective Avro parser tree is first generated from the given schema. Then, this parser tree is traversed depth-first in the given order. At each node, a parser block of the respective type is generated. As an example, Fig. 8a specifies the Avro parser tree of the schema in List. 2. Fig. 1 illustrates the parser structure generated for this example. The FSMs are initialized to schedule the parser blocks in the specified order.

## 2.2 Phase 2 – Selection and projection logic generation

The second step deals with creating the Parse, Project & Select (PPS) module based on the object parser by adding the hardware for selection and projection, given by a JSONPath expression. The JSONPath expression specifies all attributes that are relevant for projection and selection. A dollar sign at the beginning represents the root (or start) of an Avro object. Elements are separated by dots. If several attributes are required, they can be concatenated with a |. In addition, the JSONPath expression contains the selection criterion consisting of comparisons of attributes that can be combined via Boolean expressions. The JSONPath expression spans a tree, which we denote by *path tree*  $G_P(V_P, E_P)$  in the following, where the root represents the start of the Avro object and the leaves the attributes of interest. Each path in this tree also has a corresponding path in the object parser tree.

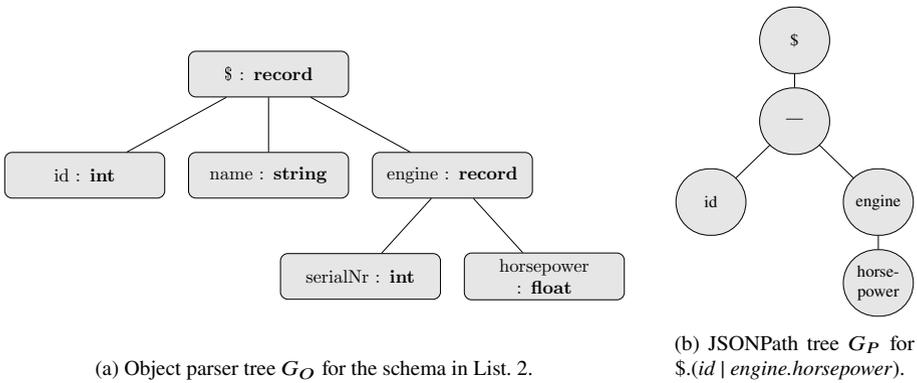


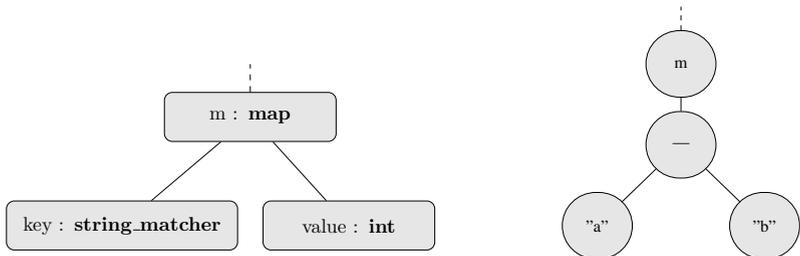
Fig. 8: Tree structures for the path evaluation.

**Extract attributes** We first consider only the projection of fields and present the specifics of selection further below. The extraction of the attributes relevant to a given query works by recursively traversing every path in the JSON path tree and at the same time determining the respective path in the parser tree. The algorithm works by starting from the roots of both trees,  $v_o \in V_O$  and  $v_p \in V_P$ . Then, for each child  $v'_p \in V_P$  of the current path tree node  $v_p$ , the corresponding child  $v'_o \in V_O$  of the current object parser tree node  $v_o$  is determined. The same procedure is repeated for the obtained pair  $(v'_p, v'_o)$  of child nodes. Once, the path tree node  $v'_p$  is a leaf in the path tree, the respective object parser tree node  $v'_o$  represents one attribute that is required by the query. Therefore, the signals *valid\_out* and *data\_out* of the parser block that was previously generated for this object parser tree node  $v'_o$  are then connected with the first stage register.

*Example:* Take as an example the parser tree  $G_O$  shown in Fig. 8a and the path tree  $G_P$  of the path expression  $$(id | engine.horsepower)$  shown in Fig. 8b. For extracting all attributes needed for selection and projection, we first consider the root nodes of both trees, that is, the record parser block of the parser tree and the  $\$$  node of the path tree. Starting from

the \$ node, we next consider the children in the path tree. In Fig. 8b, this is the | node, which in turn indicates that the path splits into two subpaths, each considering the respective child node in the path tree and its associated parser block. Thus, in the first subpath, the *id* : int parser block and the *id* node in the path tree are selected. Given that the *id* node is a leaf node, the currently selected parser block (*id* : int) is marked for extraction so that a later selection or projection can be performed. The same procedure is repeated in the right subpath. However, in this case, the observed node in the path tree (*engine*) is not a leaf node, which is why its child (*horsepower*) must be again taken into consideration. As the currently selected parser block is of record type, the corresponding parser block to the *horsepower* node can be selected (*horsepower* : float). At this point, the path node is a leaf node as well, which means that the *horsepower* block is also marked for extraction. After all paths have been traversed, the signals *out\_data* of all parser blocks marked for extraction are connected to the stage 1 registers, which are written with a one on the signal *out\_valid* of the selected block.

Attribute extraction works straightforward for elementary and record parser blocks. However, map and union types have peculiarities as discussed in the following. While with a record the children in the path tree correspond directly to the children in the parser block, the path children of the map type correspond to its queried key strings (cf. Fig. 9). Accordingly, no parser blocks are selected, but the *string\_matcher* match dictionary is set up with the searched key string. The resulting signal *key\_match* is then used as a write condition for writing *out\_data* to the projection register in stage 1. Since several entries can be extracted from a map, there is also the possibility to split the path into subpaths using | nodes, as can be seen in Fig. 9b. In this case, a separate entry is created in the *string\_matcher* dictionary for each of the visited children (here "a" and "b"), each with its independent signal *key\_match*. The signal *out\_data* of the value parser block is then connected to a separate stage 1 register for each child and only written if the respective *key\_match* is present.



(a) Object parser tree  $G_O$  for the map type.

(b) JSONPath tree  $G_P$  for path  $.m("a"|"b")$ .

Fig. 9: Tree structures for the path evaluation of the map type.

Whereas with the maps type, the same parser block can be projected multiple times, the union type contains multiple parallel parser blocks, one for each data type the field may potentially take. During the projection and selection phase, it is therefore necessary to decide which of the types and, with that, which of the parser blocks is relevant for the query

and should therefore be connected with the stage 1 register. The selection of the target type could be done at runtime or statically at design time. Since we want to enforce a fixed data layout for further data processing on the FPGA, we have opted for static types. However, since the JSONPath syntax does not provide for type casts or similar, we extended the syntax accordingly. For this purpose, the expected type is specified after the attribute name separated by two colons (e.g., for a union variable *sensor* which is to be mapped to the *int* type, the syntax is `$.sensor::union(int)`). Assuming it is not statically known which union type is to be expected, all possible types must be projected as separate fields, which of course creates overheads, but still allows to preserve a fixed layout.

**Apply selection logic** Filter expressions start with a question mark and iterate over all array entries which can be referenced via the `@` symbol. To be able to perform selections on the record scope, we have again extended the JSONPath notation lightly. Thereby, entire records can be filtered by specifying the filter expression directly at the root of the record (e.g., `[$?(id=0)]`). If the expression evaluates to false, the entire record is discarded in the parser and not passed on for further processing. The resulting path tree  $G_P$  is depicted in Fig. 10.

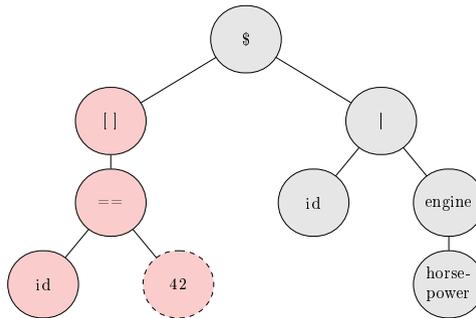


Fig. 10: Path tree  $G_P$  with selection for JSONPath `[$?(id=42)].(id | engine.horsepower)`.

The filter expression is always placed at the left child of the root node, so before traversing the path, it is checked first whether there is a filter expression at this position. If this is the case, the filter expression is evaluated first. When visiting the comparison node, the corresponding compare logic is instantiated based on the data types of its children. Here, we support direct comparisons of two strings, two booleans, or two enums and `<`, `>`, `≤`, `≥`, `==` comparisons of two integers or two floats. When referencing attributes, as is the case with the *id* block in Fig. 10, the corresponding parser block is connected to a stage 1 register as described in the previous paragraph. The register is then used as input for the comparison logic, as shown in Fig. 1 and the result is connected to the valid flag of the stage 2 register. The right subtree represents the projection paths and therefore is evaluated as described before. The stage 1 registers of the attributes to be projected must then be connected to the stage 2 registers.

### 2.3 Phase 3 – Accelerator generation

In the third phase, additional logic is generated to obtain an accelerator module that can be deployed on an FPGA. For communication with other accelerators or system interfaces, the generated PPS module is connected to AXI interfaces. Furthermore, if the Avro objects are embedded in a wrapper format, such as the container format or the wire format, its decoding must be carried out. Furthermore, since the PPS module generated in the previous phase can only process 1 byte per clock cycle, it is necessary to instantiate multiple PPS modules to run parallel in order to achieve a high throughput.

Usually, we allocate 8 parallel PPS modules, each being fed by one channel. The wire format introduced in Sect. 1.1 contains multiple subsequent buffers. Each buffer stores one or multiple Avro objects. The buffers are assigned to the channels in a round-robin fashion. As we instantiate 8 parallel channels, this scheme can saturate a 64-bit interface at a throughput of 1 byte/cycle and channel.

However, this approach also results in limitations concerning the input data. If the objects are required to be reassembled after processing in the same order they entered, each buffer may only contain one Avro. Splitting individual Avro objects to the channels and assembling from the parallel PPS modules has then to happen in the same round-robin fashion. For larger objects, this is no problem, since the overhead of the buffer length field on the overall data footprint can be neglected. Should this be a problem nevertheless, it could be solved by modifying the Wire format: By splitting the 4 bytes of the length field into 3 bytes for the buffer length and 1 byte for the number of records contained, the output stream could later be reassembled based on the given number of records in each buffer and channel.

### 2.4 Automatic hardware generation

Automatic generation of hardware is typically done by emitting VHDL or Verilog code, using template engines, or even worse, using a large number of print statements in the generator. This results in extremely poor readability and maintainability of the generator code. We therefore decided to generate all logic circuits using Python-based HDL called Amaranth<sup>5</sup>. In Amaranth, hardware is described at the register transfer level as in VHDL or Verilog, while allowing for modern language features of Python such as object orientation. This is especially advantageous for the implementation of the parser blocks, since, for example, all complex parser blocks can inherit from a class `sequential_parser`, which already contains basic FSM logic for sequential activation of the instantiated parser blocks. In addition, the interface for parser blocks introduced in Sect. 2.1 can be inherited by all blocks via an abstract class, so instantiating parser blocks in other blocks (e.g., in the record parser) can be easily implemented using attributes of the abstract class type. Moreover, the object structure created in this way is perfectly suited for traversing the parser tree

<sup>5</sup> Amaranth HDL: <https://github.com/amaranth-lang/amaranth>

(see Sect. 2.2). The described hardware can then be finally output in Verilog, allowing the generated modules to be used platform-independently. The Python-based HDL not only allows us to create highly nested modules, but it is also possible to leverage well-established Python modules for parsing the Avro schema and JSONPath expressions.

### 3 Evaluation

We selected two stream processing benchmarks to evaluate our approach and the generated circuits. These are the Yahoo Streaming Benchmark [Ch16], which monitors advertising campaigns, and the RIOTBench [SCS17], which includes various applications for the Internet of Things. In both cases, we only consider the parsing stage, as well as subsequent projections and selections for our evaluation. As both benchmarks originally expect JSON-formatted data as input, an equivalent Avro schema had to be defined first. However, to define these schemas, we first discuss the input data of both benchmarks. Following this, a path expression is to be chosen based on the selection & projection applied in each benchmark.

**Yahoo Streaming Benchmark** The input JSON data of the Yahoo Streaming benchmark consists of 7 string attributes. These are first three UUIDs (*user\_id*, *page\_id* & *ad\_id*) to identify the advertisement event, two type fields (*ad\_type* & *event\_type*), a timestamp (*event\_time*) and the IP address of the user (*ip\_address*). We decided to encode the UUIDs using the fixed type (16 Byte), the type fields (*ad\_type* & *event\_type*) using one enum in each case, the timestamp using a long and the IP address using a string. During selection, it is tested whether the *event\_type* enum type is set to the "view" enum element<sup>6</sup> Subsequently, the attributes *ad\_id* and *event\_time* are projected in the benchmark, which results in the following path expression: `[$[?event_type = 'view'].(ad_id | event_time )`.

**RIOTBench** Similarly to the Yahoo Benchmark, the RIOTBench originally works with JSON formatted data. The structure of the JSON data is based on the SenML format, which is used as an exchange format for sensor measurements. The JSON records received in the benchmark are encoded as a JSON record with initially two fields. This is first a *timestamp* of the measurements and second an *array* which contains all measured values. The array is in turn always comprised of 8 records. Each measurement records contain three fields, a *value* field with the actual measured value, a field for the *name* of the measured value, and a field for the *physical unit*. While the name and unit fields are encoded as a string, the value field can be encoded either as an integer or as a float depending on the physical unit of the sensor measurement.

We adapted the benchmark for Avro. The Avro schema first contains a long timestamp and a field of type map (*values*), which contains the sensor measurements. We chose to use a map

---

<sup>6</sup> In the original benchmark, this is a string comparison as the *event\_type* attribute is formatted as a string.

for the measurements instead of an array, as this makes it easier to extract its entries. A map entry, respectively a sensor measurement, consists of the sensor name as key and union type for its value. The union type, in turn, defines one type for each physical unit. Accordingly, the correct value type (int or float) can be defined for each physical unit. In the SmartCity query of the RIoTbench, five variables are initially projected after parsing. All five variables are then used for a selection expression. The resulting path expression can be seen in List. 3.

```

$[?values.temperature::union(senml_fahrenheit) >= -12.5
  & values.temperature::union(senml_fahrenheit) <= 43.1
  & values.humidity::union(senml_percentage) >= 10.7
  & values.humidity::union(senml_percentage) <= 95.2
  & values.light::union(senml_percentage) >= 1345
  & values.light::union(senml_percentage) <= 26282
  & values.dust::union(senml_percentage) >= 186.61
  & values.dust::union(senml_percentage) <= 5188.21
  & values.airquality_raw::union(senml_percentage) >= 17
  & values.airquality_raw::union(senml_percentage) <= 363]
  .(values.temperature::union(senml_fahrenheit))
  | (values.humidity::union(senml_percentage))
  | (values.light::union(senml_percentage))
  | (values.dust::union(senml_percentage))
  | (values.airquality_raw::union(senml_percentage))

```

List. 3: Path expression for the RIoTbench SmartCity query.

**System architecture** The generated parser accelerators have been evaluated on a Xilinx ZCU106 Zynq SoC. Fig. 11 depicts the architecture of our system, based on [Be19], consisting of a tightly coupled ARM CPU and programmable logic (PL). The PL contains several dynamically Reconfigurable Regions (RRs), which are connected to each other and to various interfaces via a crossbar. In each of the RRs resides a DMA engine, managed by the on-chip ARM CPU.

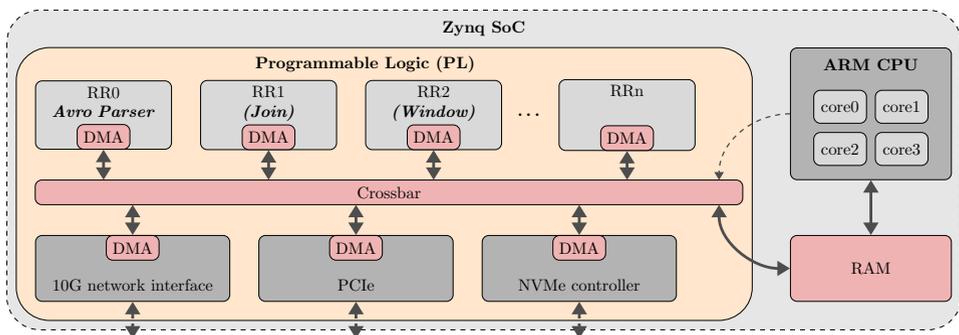


Fig. 11: FPGA-based system architecture for evaluation.

In our experiments, 3.6 MB of Avro data from the Yahoo benchmark, as well as 1.5 MB from the RiotBench, were preloaded into RAM and transferred to the parser accelerator using DMA. The results containing the parsed Avro objects were again written back to RAM via DMA. For the experiments, the entire system was clocked at 200 MHz, thus with a word width of 64 bits, a throughput of 1.6 GB/s should ideally be achievable. In practice, however, we only achieved a throughput of 1.45 GB/s, which was due to the CPU not being able to schedule new DMA descriptors fast enough. In the future, this problem could be solved by using more cores to schedule the DMAs, or even by adding a dedicated hardware component for scheduling the DMAs. For us, however, the achieved throughput is sufficient since it suffices to process the incoming data from a 10 GBit/s network interface at line rate. Our system architecture can thus be used to process and accelerate further stages or even an entire data stream processing application on the FPGA.

*Example:* Consider the Yahoo Benchmark again. First, a data stream of Avro objects is received at the network interface. This stream is then forwarded as described to the first Reconfigurable Region (RR) which contains our generated Avro parser accelerator, directly performing the first three steps of the Yahoo benchmark (parse, project & select). Then the stream of parsed, projected, and filtered Avro objects is passed to the next RR, which performs a join against a document store in the Yahoo benchmark. The tuples of the document store required for joining are also transferred via DMA from the NVMe controller to the corresponding RR. The joined tuples are then passed again to another RR to aggregate via a window function in the last step. The accelerators for processing joins and windows can be generated as shown in past research [MTA09; MTA10; TM11]. The output stream of the window is again to be stored in the document store and must be accordingly transferred back to the NVMe controller.

**Benchmark results** Besides the above experiments, we determined the maximum achievable clock frequency as well as the resource consumption of the generated accelerator engine for both benchmarks. The results are depicted in Tab. 2. The number of LUTs required for the more complex RIOTBench schema is slightly higher than for the Yahoo benchmark, but remains low overall for both accelerators, thus allowing resources to be used for further query processing, as described above. For the same reason, the maximum achieved clock frequency is also higher for the Yahoo benchmark. If the two generated accelerators are operated at their maximum clock rate, they can theoretically achieve throughputs of 3.4 GB/s (Yahoo) and 2.8 GB/s (RIOTBench). Unlike JSON and CBOR, in Avro it is likely that decoded objects have a larger data footprint at the output than at the input, making the output interface potentially the bottleneck. However, since we perform additional selections and projections in both evaluated benchmarks, the amount of data at the output is typically greatly reduced compared to the input, so the input interface remains practically the bottleneck, meaning that the given throughput numbers still correspond to the parsing speed.

Finally, the two parsing benchmarks were run with the C++ Apache Avro parser on an Intel(R) Core(TM) i7-3770 CPU to obtain an x86 CPU baseline. With one thread, a

throughput of 390 MB/s was achieved for the Yahoo benchmark and a throughput of 96 MB/s for the more complex schema of the RIOTBench. A speedup of up to 4 can be achieved by parallelizing on multiple threads for exploiting the 8 hyperthreads provided by the CPU. Here, the limited scaling results from additional overhead due to the memory management for distribution of the data as well as the synchronization of the threads. Only the parsing itself was measured in the CPU benchmarks. If selection and projections are additionally performed and the results of the threads are merged, the throughput would even be worse. In contrast, the proposed FPGA design could ideally achieve a speedup of 8.8 for the Yahoo benchmark or 29.4 for the more complex RIOTBench given that the problems of the DMA scheduling are resolved, while only requiring a minimal share of its resources.

Tab. 2: Resource consumption, clock frequency, and throughput results for two benchmarks.

|                       | benchmark               | Yahoo            | RIOTBench         |
|-----------------------|-------------------------|------------------|-------------------|
| resources (% of FPGA) | LUTs                    | 4,900 (2.1%)     | 6,691 (2.9%)      |
|                       | FFs                     | 7,288 (1.6%)     | 7,173 (1.6%)      |
|                       | maximal clock frequency | 430 MHz          | 359 MHz           |
| throughput (speedup)  | CPU single thread       | 390 MB/s (1)     | 96 MB/s (1)       |
|                       | CPU multi thread        | 1,405 MB/s (3.6) | 380 MB/s (4.0)    |
|                       | FPGA theoretical        | 3,440 MB/s (8.8) | 2,827 MB/s (29.4) |
|                       | FPGA experimental       | 1,450 MB/s (3.7) | 1,450 MB/s (15.1) |

## 4 Conclusion & Future Work

Avro's simple encoding can be interpreted using basic finite-state machines, making the parsing process perfectly suited for acceleration in hardware using FPGAs. The accelerators generated by the presented generator can achieve significant speedups compared to CPU-based parsers, although only a minimal share of the FPGA resources is required. Moreover, path expressions can be used to parse the received objects into a fixed data layout and reduce the amount of output data to avoid unnecessary data movement. The enforced data layout is then perfectly tailored to accelerate further steps of the given application on the available FPGA resources.

Optimization potential for our approach arises from the fact that parser blocks must be instantiated multiple times when the same Avro type occurs multiple times in the schema. Furthermore, the generated parser is only able to parse the schema it was generated with, making schema evolution only possible by creating and instantiating multiple parser accelerators. In the future, we want to solve these two problems by coordinating the parser blocks via an instruction set. The schema would then be translated into a program that would control the sequence of parser blocks. Thus, different schemas and, accordingly, schema evolution could be achieved simply by executing different programs.

## References

- [Ap13] Apache Software Foundation: Apache ORC Specification v1, Oct. 13, 2013, URL: <https://orc.apache.org/specification/>.
- [Ap21] Apache Software Foundation: Apache Avro 1.11.0 Specification, Oct. 29, 2021, URL: <https://avro.apache.org/docs/1.11.0/spec.pdf>.
- [Ap22a] Apache Software Foundation: Apache Arrow Columnar Format Specification v1, Dec. 1, 2022, URL: <https://arrow.apache.org/docs/format/Columnar.html>, visited on: 12/01/2022.
- [Ap22b] Apache Software Foundation: Apache Parquet Specification, Mar. 24, 2022, URL: <https://parquet.apache.org/docs/>.
- [Be19] Becher, A.; Herrmann, A.; Wildermann, S.; Teich, J.: ReProVide: Towards Utilizing Heterogeneous Partially Reconfigurable Architectures for Near-Memory Data Processing. In: Gesellschaft für Informatik, Bonn, 2019, URL: <https://doi.org/10.18420/btw2019-ws-04>.
- [BH20] Bormann, C.; Hoffman, P.: Concise Binary Object Representation (CBOR), STD 94, RFC Editor, Dec. 2020, URL: <https://tools.ietf.org/pdf/rfc8949>.
- [Br17] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format, RFC 8259, Dec. 2017, URL: <https://www.rfc-editor.org/info/rfc8259>.
- [Ch16] Chintapalli, S.; Dagit, D.; Evans, B.; Farivar, R.; Graves, T.; Holderbaugh, M.; Liu, Z.; Nusbaum, K.; Patil, K.; Peng, B. J.; Poulosky, P.: Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Pp. 1789–1792, 2016, URL: <https://doi.org/10.1109/IPDPSW.2016.138>.
- [Da22] Dann, J.; Wagner, R.; Ritter, D.; Faerber, C.; Froening, H.: PipeJSON: Parsing JSON at Line Speed on FPGAs. In: DaMoN'22, 2022, URL: <https://doi.org/10.1145/3533737.3535094>.
- [El10] El-Hassan, F.; Ionescu, D.: A hardware architecture of an XML/XPath broker for content-based publish/subscribe systems. In: 2010 International Conference on Reconfigurable Computing and FPGAs. IEEE, pp. 138–143, 2010.
- [Gö07] Gössner, S.: JSONPath - XPath for JSON, Feb. 20, 2007, URL: <https://goessner.net/articles/JsonPath/>.
- [Go22] Google Inc.: Protocol Buffers Version 3 Language Specification, Nov. 1, 2022, URL: <https://developers.google.com/protocol-buffers/>.
- [Ha22] Hahn, T.; Becher, A.; Wildermann, S.; Teich, J.: Raw Filtering of JSON data on FPGAs. In: DATE'22, Antwerpen. Mar. 14–23, 2022, URL: <https://doi.org/10.23919/DATE54114.2022.9774696>.

- [HWT22] Hahn, T.; Wildermann, S.; Teich, J.: Auto-Tuning of Raw Filters for FPGAs. In: FPL'22, Belfast, United Kingdom. Aug. 29–Sept. 2, 2022.
- [KSS08] Koch, C.; Scherzinger, S.; Schmidt, M.: XML Prefiltering as a String Matching Problem. In: ICDE'08. Pp. 626–635, 2008, URL: <https://doi.org/10.1109/ICDE.2008.4497471>.
- [Li17] Li, Y.; Katsipoulakis, N. R.; Chandramouli, B.; Goldstein, J.; Kossmann, D.: Mison: A Fast JSON Parser for Data Analytics./, 2017, URL: <https://doi.org/10.14778/3115404.3115416>.
- [Me10] Melnik, S.; Gubarev, A.; Long, J. J.; Romer, G.; Shivakumar, S.; Tolton, M.; Vasilakis, T.: Dremel: Interactive Analysis of Web-Scale Datasets. In: VLDB'10. 2010, URL: <http://www.vldb2010.org/accept.htm>.
- [Mi09] Mitra, A.; Vieira, M.; Bakalov, P.; Najjar, W.; Tsostras, V.: Boosting XML Filtering with a Scalable FPGA-based Architecture./, 2009, URL: <https://arxiv.org/abs/0909.1781>.
- [MTA09] Mueller, R.; Teubner, J.; Alonso, G.: Streams on Wires: A Query Compiler for FPGAs./, Aug. 2009, URL: <https://doi.org/10.14778/1687627.1687654>.
- [MTA10] Mueller, R.; Teubner, J.; Alonso, G.: Glacier: a query-to-hardware compiler. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. Pp. 1159–1162, 2010, URL: <https://doi.org/10.1145/1807167.1807307>.
- [Pe21] Peltenburg, J.; Hadnagy, Á.; Brobbel, M.; Morrow, R.; Al-Ars, Z.: Tens of gigabytes per second JSON-to-Arrow conversion with FPGA accelerators. In: ICFPT'21. 2021, URL: <https://doi.org/10.1109/ICFPT52863.2021.9609833>.
- [Sa12] Sadoghi, M.; Javed, R.; Tarafdar, N.; Singh, H.; Palaniappan, R.; Jacobsen, H.-A.: Multi-query Stream Processing on FPGAs. In: 2012 IEEE 28th International Conference on Data Engineering. Pp. 1229–1232, 2012, URL: <https://doi.org/10.1109/ICDE.2012.39>.
- [SCS17] Shukla, A.; Chaturvedi, S.; Simmhan, Y.: RIoTBench: An IoT benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience*, Oct. 2017, URL: <https://doi.org/10.1002%2Fcpe.4257>.
- [TM11] Teubner, J.; Mueller, R.: How Soccer Players Would Do Stream Joins. In: SIGMOD '11, Athens, Greece, 2011, URL: <https://doi.org/10.1145/1989323.1989389>.
- [TWN12] Teubner, J.; Woods, L.; Nie, C.: Skeleton Automata for FPGAs: Reconfiguring without Reconstructing. In: SIGMOD '12, 2012, URL: <https://doi.org/10.1145/2213836.2213863>.
- [WA11] Woods, L.; Alonso, G.: Fast data analytics with FPGAs. In: 2011 IEEE 27th International Conference on Data Engineering Workshops. IEEE, pp. 296–299, 2011, URL: <https://doi.org/10.1109/ICDEW.2011.5767669>.



# Working with Disaggregated Systems. What are the Challenges and Opportunities of RDMA and CXL?

Andreas Geyer,<sup>1</sup> Daniel Ritter,<sup>2</sup> Donghun Lee,<sup>3</sup> Minseon Ahn,<sup>3</sup> Johannes Pietrzyk,<sup>1</sup>  
Alexander Krause,<sup>1</sup> Dirk Habich,<sup>1</sup> Wolfgang Lehner<sup>1</sup>

**Keywords:** RDMA; CXL; Disaggregated Systems; Network; Memory

## 1 Motivation

The usage of disaggregated systems in large scale data-centers offers a lot of flexibility and easy scalability in comparison to the traditional statically configured scale-up and scale-out systems. Disaggregated architectures allow for the creation of software composable systems [Li17, Li18]. On the one hand, this allows seamless integration of specialized hardware like FPGAs or GPUs as well as a high degree of elasticity to scale a system with its workload by dynamically adding and removing resources via software. On the other hand, however, it also brings several challenges like an additional communication overhead for memory accesses, which is especially critical for In-Memory databases.

With the more traditional scale-up system approach, all communication happens on the same machine and is – depending on the interconnect and support by specific hardware components [Yu07] – reasonably fast. To access data from main-memory or storage, it is only necessary to retrieve it from the directly attached hardware. With growing systems, the multi-socket system became the de facto standard. This introduced the challenge of *non-uniform memory access (NUMA)* [Ps16]. The larger distance to hardware on the same machine but connected to another socket introduces a latency overhead when collecting data from this hardware. With growing NUMA distance to the desired hardware, the latency overhead grows for each communication. Therefore, such systems are designed to keep the NUMA distances as small as possible, which leads to the Near-Memory Processing (NMP) or compute paradigm [Ps16, Ki14, Pa10]. Modern software has already mostly adapted to the challenges NUMA entails, but it is still a research field of its own.

In comparison to these traditional system approaches, the usage of disaggregated systems in large scale data-centers offer a lot of flexibility and easy scalability. It allows for the

---

<sup>1</sup> Technische Universität Dresden, Database Systems Group, Nöthnitzer Straße 46, 01187 Dresden, Germany, {first.last}@tu-dresden.de

<sup>2</sup> SAP SE, Germany, daniel.ritter@sap.com

<sup>3</sup> SAP Labs, Korea, [dong.hun.lee|minseon.ahn}@sap.com

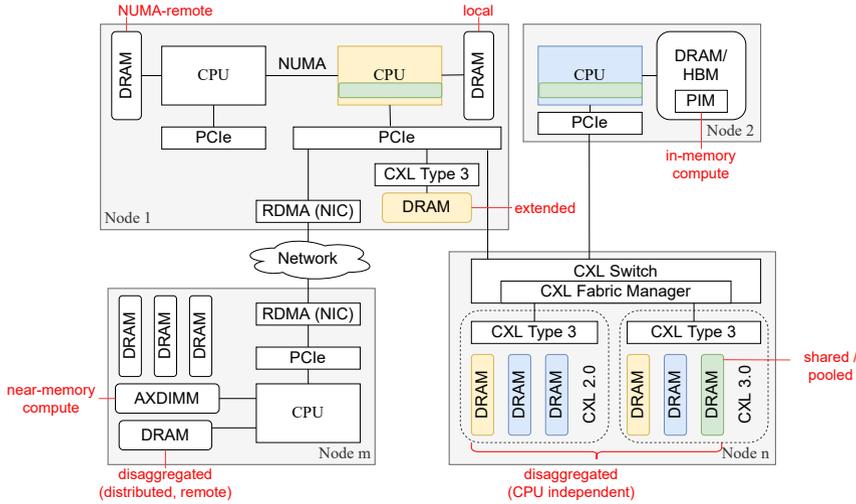


Fig. 1: Disaggregated Memory Approaches

creation of software composable systems [Wa22]. Therefore, the utilization of the available hardware can be a lot better than in scale-up or scale-out scenarios. If memory or CPU is not needed anymore, it is returned to the pool and can immediately be reassigned for other tasks. However, composing the hardware through any virtualization leads to physically distributed hardware and thus the inevitable necessity to cope with NUMA effects. Traditionally, performance optimization was achieved through either copying the data to the execution unit or moving the function to the data. This communication is a potential bottleneck for fast computation even though fast communication techniques like *Remote Direct Memory Access (RDMA)* and *Compute Express Link (CXL)* exist.

## 2 Classification

From a database perspective, data and its locality has always been the center of attention. Correctly deciding on which data to prefetch from disk into memory was crucial for high performance. However, hardware in the cloud setting advanced over the years from scale-up and scale-out servers to disaggregated systems. Thus, imposing an even larger design space to consider for data placement. With our research, we want to investigate the implications of different hardware or memory extension techniques, based on RDMA and CXL. To achieve that, we introduce a classification of the different memory distances, based on their actual physical distance but taking the transport layer into account.

Figure 1 highlights the multitude of potential distances that can occur when scale-up servers meet hardware disaggregation. A single compute node can consist of multiple sockets, where a socket can then be further expanded both via RDMA, e.g. over InfiniBand, and

CXL-based switches. Hence, we first divide memory into *local* and *NUMA-remote*, which is meant to physically reside in the same server. *Extended memory* describes CXL memory devices, which are directly attached via CXL through the PCIe connector of the mainboard. Devices, that are attached to a socket through a CXL switch are called *disaggregated memory* (DM). With CXL 3.0 it will be possible to share this DM between multiple systems. When RDMA is used to connect two systems, we consider such memory to be *disaggregated remote memory* (DRM). The distinction between DM and DRM is made because the DM is managed from the host on the *active* side. DRM, however, is managed by another host, which is just exposing a part of its own memory to remote systems. Additionally, technologies like *high bandwidth memory* (HBM) [Ju17] and *processing in memory* (PIM) [GHI95] can be used and further increase the performance of the system.

### 3 Use Cases

Recent research has already outlined the importance of available memory for in-memory database systems [Ah22]. Without claiming completeness, we see a couple of main use cases for database centric systems that are necessary to consider when working with hardware disaggregation.

First, dynamic memory expansion for *In-Memory Database Management Systems* (IMDBMS) through CXL. Even when data keeps increasing and the DIMM slots in the server are already full, the customer can expand the memory space without upgrading their server nor memory devices for scale-up. For this, the IMDBMS may allocate the operational memory in the expanded memory without any change in data placement or may move the table data to the expanded memory. The latter one would be more beneficial as table data accesses consist of lots of sequential accesses than operational memory in general and the longer latency of expanded memory can be hidden by prefetching.

Second, integrating memory expansion for multiple sockets of a single server through CXL 2.0, since it allows the connection of multiple CXL devices through a CXL switch. Attaching only one socket in a server to a CXL switch will cause NUMA-like effects among the sockets. Providing a single memory pool, which can be accessed by every socket of the server will contribute to even access latency with better bandwidth. CXL 2.0 does not allow full sharing of a whole memory device, hence we would allocate dedicated regions per socket. A limiting factor of the effectiveness for this scenario is the hardware itself: currently, we would need the same amount of wired connections between the memory pool and each socket of the server. With more servers and hence more sockets in the joint, concurrency becomes an even more serious issue and dedicated memory areas are an important building block for multi-server memory pools. Further, the amount of required wired connections scales linearly to the amount of sockets of all servers.

Third, a shared memory pool among multiple servers, but through CXL 3.0 and hence the ability of sharing the pool as a whole. With the newest CXL standard, even the same memory

regions on a device can be shared among multiple hosts. This new degree of freedom requires precise rights management, i.e. read-/write-permissions and data ownership.

Fourth, with PIM and dedicated RDMA-connected servers enabling the offloading of some operators to the data. Due to computational power near the data this could reduce the data transfer significantly. Therefore, when grouping the data access like in [Ge23] it would be possible to reduce the latency and interleave data transfer and computation.

Based on these use cases, we see the possibility to further increase the systems complexity by linking these multiple servers also with InfiniBand. Such a setup would allow to limit the amount of CXL wires by replacing them with less InfiniBand cables. In this configuration, one server could serve as a memory managing unit, which exposes selected data regions to the other servers.

## 4 Call to Action

Based on our classification and the presented use cases, we identify a set of promising research directions. Traditionally, data was either shipped to the central processing unit (CPU) or the processing function or operator was moved to the data. We argue that with the rise of high performance interconnects like RDMA and CXL, there is no such black-and-white decision anymore. Our research focuses on investigating the performance implications of the different memory categories, based on our classification from Section 2, from the perspective of an IMDBMS. We already conducted initial experiments for *disaggregated remote memory* [Ge23] and want to extend our prototype by combining the different memory classes following our use case description of Section 3. The scope of our endeavor is not limited to experimenting with memory extensions, but also on how to include computational storage or Processing In memory (PIM) in such a system.

Conceptually speaking, we want to build a database prototype, that can leverage different kinds of attached memory, based on a suitable abstraction layer. Our research aims to emit three main contributions: (1) define and confirm the different memory classes, based on our experiments with the combination of RDMA and CXL attached memory. (2) we will provide an analytical model on when to use which memory. That includes the decision of when to ship the data, e.g. because of computational limitations and when operators should be offloaded. Lastly, (3) we will identify a set of common access primitives, that can be leveraged to work with all memory classes through a dedicated API.

We would be delighted to present our ideas and the memory disaggregation classification at the workshop and discuss the presented ideas. The valuable feedback of the attendees will help us to further refine our classification both in terms of preciseness and applicability.

**Acknowledgements.** We would like to thank Marcel Weisgut from Hasso-Plattner-Institut, Potsdam for the fruitful discussions on CXL and his contributions to the memory disaggregation classification.

## Bibliography

- [Ah22] Ahn, Minseon; Chang, Andrew; Lee, Donghun; Gim, Jongmin; Kim, Jungmin; Jung, Jaemin; Rebholz, Oliver; Pham, Vincent; Malladi, Krishna T.; Ki, Yang-Seok: Enabling CXL Memory Expansion for In-Memory Database Management Systems. In: DaMoN. ACM, pp. 8:1–8:5, 2022.
- [Ge23] Geyer, Andreas; Krause, Alexander; Habich, Dirk; Lehner, Wolfgang: Pipeline Group Optimization on Disaggregated Systems. In: CIDR. 2023. to appear.
- [GHI95] Gokhale, M.; Holmes, B.; Iobst, K.: Processing in memory: the Terasys massively parallel PIM array. *Computer*, 28(4):23–31, 1995.
- [Ju17] Jun, Hongshin; Cho, Jinhee; Lee, Kangseol; Son, Ho-Young; Kim, Kwiwook; Jin, Hanho; Kim, Keith: HBM (High Bandwidth Memory) DRAM Technology and Architecture. In: 2017 IEEE International Memory Workshop (IMW). pp. 1–4, 2017.
- [Ki14] Kissinger, Thomas; Kiefer, Tim; Schlegel, Benjamin; Habich, Dirk; Molka, Daniel; Lehner, Wolfgang: ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workload. In (Bordawekar, Rajesh; Lahiri, Tirthankar; Gedik, Bugra; Lang, Christian A., eds): International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2014, Hangzhou, China, September 1, 2014. pp. 74–85, 2014.
- [Li17] Li, Chung-Sheng; Franke, Hubertus; Parris, Colin; Abali, Bülent; Kesavan, Mukil; Chang, Victor I.: Composable architecture for rack scale big data computing. *Future Gener. Comput. Syst.*, 67:180–193, 2017.
- [Li18] Lin, An-Dee; Li, Chung-Sheng; Liao, Wanjiun; Franke, Hubertus: Capacity Optimization for Resource Pooling in Virtualized Data Centers with Composable Systems. *IEEE Trans. Parallel Distributed Syst.*, 29(2):324–337, 2018.
- [Pa10] Pandis, Ippokratis; Johnson, Ryan; Hardavellas, Nikos; Ailamaki, Anastasia: Data-Oriented Transaction Execution. *Proc. VLDB Endow.*, 3(1):928–939, 2010.
- [Ps16] Psaroudakis, Iraklis; Scheuer, Tobias; May, Norman; Sellami, Abdelkader; Ailamaki, Anastasia: Adaptive NUMA-aware data placement and task scheduling for analytical workloads in main-memory column-stores. *Proc. VLDB Endow.*, 10(2):37–48, 2016.
- [Wa22] Wang, Ruihong; Wang, Jianguo; Idreos, Stratos; Özsu, M. Tamer; Aref, Walid G.: The Case for Distributed Shared-Memory Databases with RDMA-Enabled Memory Disaggregation. CoRR, abs/2207.03027, 2022.
- [Yu07] Yu, Hao; Moreira, José E.; Dube, Parijat; Chung, I-Hsin; Zhang, Li: Performance Studies of a WebSphere Application, Trade, in Scale-out and Scale-up Environments. In: 21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA. IEEE, pp. 1–8, 2007.



# What We Can Learn from Persistent Memory for CXL

Lawrence Benson<sup>1</sup>, Marcel Weisgut<sup>1</sup>, Tilmann Rabl<sup>1</sup>

## Abstract:

With high-capacity Persistent Memory (PMem) entering the long-established data center memory hierarchy, various assumptions about the performance and granularity of memory access have been disrupted. To adapt existing applications and design new systems, research focused on how to efficiently move data between different types of memory, how to handle varying access latency, and how to trade off price for performance. Even though Optane is now discontinued, we expect that the insights gained from previous PMem research apply to future work on Compute Express Link (CXL) attached memory. In this paper, we discuss how limited hardware availability impacts the performance generalization of new designs, how existing CPU components are not adapted towards different access characteristics, and how multi-tier memory setups offer different price-performance trade-offs. To support future CXL research in each of these areas, we discuss how our insights apply to CXL and which problems researchers may encounter along the way.

## 1 Introduction

With the arrival of Intel Optane Persistent Memory (PMem) in 2019, research on new data management techniques for byte-addressable persistent memory increased significantly. Among other questions, this research investigates how to handle varying memory access latency, how to place data based on available capacity, and how to design for memory access sizes larger than a single cache line but smaller than a page [BMR21; Lu20; Re18]. However, in 2022, Intel announced that their Optane product line will be discontinued in favor of recent trends toward Compute Express Link (CXL) [GZ22]. We expect that while Optane was abandoned, research based on it still provides valuable insights.

In light of Intel citing CXL as one of the reasons for ending Optane, in this paper, we raise the question: “What can we learn from PMem research for future CXL research?” Based on benchmarks that we conducted in previous work on PerMA-Bench [BPR22], a configurable benchmark framework for PMem access, we look at three insights from PMem that also apply to future research on CXL.

First, we discuss how limited hardware access can lead to solutions that are too specialized for one hardware configuration or not specialized enough. We then discuss how existing CPU components interact with new memory types, based on the prefetching behavior with PMem. Finally, we show that different memory types offer different price-performance trade-offs depending on the use case. Even though CXL-attached memory is not yet generally available, these insights highlight some challenges that future research faces when integrating new memory types into a long-established memory hierarchy.

---

<sup>1</sup> Hasso Plattner Insitut, Universität Potsdam, Germany. {first.last}@hpi.de

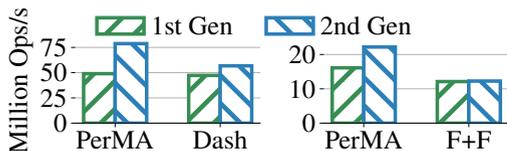


Fig. 1: Performance of PerMA and actual index implementation of Dash and FAST+FAIR.

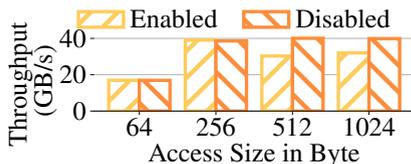


Fig. 2: Impact of prefetcher on PMem random read bandwidth.

## 2 Transferring Insights from PMem to CXL-Attached Memory

In this section, we discuss how insights derived from PMem transfer to future CXL research.

**Persistent Memory.**<sup>2</sup> PMem can be used as a volatile DRAM extension (*Memory Mode*) or explicitly as PMem beside DRAM (*App Direct Mode*). When using PMem in App Direct Mode, PMem and DRAM share the application’s unified virtual address space, i.e., data stored in both types of memory can be prefetched by the CPU. Data in PMem is accessed via load/store instructions issued in the CPU. Through a modified DDR-4 interface (called DDR-T), the CPU communicates with PMem DIMMs in 64 Byte cache lines. However, Optane’s internal media access occurs at 256 Byte, causing read and write amplification for access smaller than 256 Byte. PMem’s access latency for random reads and writes from and to PMem is  $\sim 2\text{-}5\times$  higher than DRAM’s and read/write bandwidth is  $\sim 2.5/5\times$  lower. Thus, even though DRAM and PMem share the same interface, applications designed for PMem have to account for its worse performance. As future CXL-attached memory will have higher latency and, for CXL versions 1.1 and 2.0, lower bandwidth than CPU-attached memory (bound by PCIe 5.0 compared to memory channels), CXL designs face similar issues as PMem designs.

**Benchmark Setup.** We evaluate two servers with 256 GB PMem DIMMs of the first and second-generation Optane. The first generation server contains a Cascade Lake CPU with 18 cores and six PMem DIMMs at 2933 MT/s. The second generation server contains an Ice Lake CPU with 32 cores and eight PMem DIMMs at 3200 MT/s. Both systems run Ubuntu 20.04 with a 5.4 kernel.

**Application Tailoring.** To understand how well applications utilize the hardware, we compare the `lookup()` performance of PMem index structures modeled in PerMA-Bench with the actual index implementations. The results obtained with PerMA-Bench show a performance upper-bound, as they include only memory access without computation or branching logic. The results for the hash index Dash [Lu20] and the B-Tree index FAST+FAIR [Hw18] are shown in Figure 1. The memory access for Dash is modeled as a 512 Byte random read, as Dash reads two consecutive 256 Byte hash buckets to find an entry. For FAST+FAIR, we issue  $3\times$  random 512 Byte reads that represent B-Tree node lookups. The experiments are run with 16 threads. We observe that while the underlying bandwidth improves across generations, the indexes do not (fully) utilize this. Unlike on

<sup>2</sup> For a more in-depth introduction to PMem, we refer to [BPR22].

the first generation server, Dash spends ~20% of all cycles on non-memory access on the second generation server, which contains more PMem DIMMs, a newer CPU, and better DRAM. Due to the high price of Optane, researchers often do not have access to different setups, which leads to tailoring the application towards only a single setup and, in turn, does not always generalize. On the other hand, we see FAST+FAIR as an index designed for general PMem before Optane became available. The improved memory bandwidth does not translate to the index, as FAST+FAIR spends a lot of time on heavy-weight locking and inefficient PMem access on patterns. As FAST+FAIR was designed pre-Optane, we see that the system is not tailored enough to the underlying hardware, and performance is lost.

*Insight 1:* Due to limited hardware availability, systems are tailored too much toward a single setup or not tailored enough toward the actual hardware. Through the CXL abstraction, future systems will cover a wider range of memory performance characteristics. Thus, it is important to design and research robust systems that generalize across different hardware and multiple memory tiers.

**Prefetching.** As a new layer in the long-established memory hierarchy, it is important to understand how well PMem interacts with existing CPU components, which are optimized for caches and DRAM. In Figure 2, we show the impact of the hardware prefetchers for random memory reads on the second-generation Optane server. We en-/disable all hardware prefetchers and run on 16 threads. We see that for small access sizes (< 256 Byte), the prefetcher does not impact performance, i.e., the prefetcher does not prefetch. However, for 512 and 1024 Byte access, the prefetcher speculatively loads unnecessary data not accessed by the user in the background, reducing the available bandwidth for requested reads. We observe this in hardware performance counters, where the underlying bandwidth utilization is identical in both runs, but the effective bandwidth in the application is not. Thus, disabling the prefetcher actually improves performance in this case. This effect is also observable for 2048 Byte access but not for 4096 Byte or more [Da21], as page-size access is a well-understood and optimized pattern in DRAM. As Optane’s internal access occurs at 256 Byte granularity, most applications design access in multiples of 256 Byte. As a consequence, a 512 Byte random access to Optane, e.g., a node lookup in FAST+FAIR, spans only two Optane “cache lines”, which should not trigger prefetching. However, the prefetcher views these 512 Byte as regular DRAM access, spanning eight consecutive cache lines, and starts prefetching for sequential access.

*Insight 2:* Prefetchers are highly optimized toward 64 Byte DRAM cache line access, and CXL specifies 64 Byte transfers in the transaction layer [Co22, p. 167]. However, CXL abstracts from the underlying device, i.e., it could support Optane PMem or other memory devices, and memory behind CXL may not be accessible in 64 Byte granularity. As all CPU- and CXL-attached memory is available in the same unified virtual address space, prefetchers operate on both types of memory. Future research should investigate how existing components, like the prefetcher, interact with memory that is not attached directly to the CPU and has different access characteristics. However, unlike Insight 1, this cannot be solved by applications alone and most likely requires hardware changes as well.

|      | €/GB capacity | seq. read   | rnd. read   | seq. write  | rnd. write  |
|------|---------------|-------------|-------------|-------------|-------------|
| PMem | <b>12.77</b>  | <b>0.22</b> | <b>0.33</b> | <b>0.60</b> | 2.12        |
| DRAM | 59.37         | 0.38        | 0.46        | 0.70        | <b>0.91</b> |

Tab. 1: Price-performance of PMem and DRAM. Read/write values in €/GB/s. Calculation based on listing prices from dell.de in February 2022.

**Price-Performance.** In Table 1, we show the price-performance for basic sequential/random read/write access in PMem and DRAM on the same second-generation Optane server while disregarding persistence. The data access-related prices per GB of throughput are normalized to the device’s price per GB to avoid including the higher price for larger capacity. We see that the price per GB capacity is significantly lower for the PMem DIMMs than for the high-end DRAM DIMMs. As PMem is not available in cloud vendors, we base our calculations on the list price on dell.de [De22] in February 2022. Focusing on the relative scale between the listed prices rather than on the exact monetary values, for sequential access and random reads, we observe that PMem has a better price-performance ratio, as the bandwidth is often only 2–3× worse while the price per GB capacity is about 5× better. DRAM outperforms PMem only for 64 Byte random writes, where PMem bandwidth is very low because of high write amplification.

*Insight 3:* For applications that do not require peak performance or persistence, PMem can be used as a cheaper DRAM alternative with significantly higher capacity. As increasing memory capacity is a selling point of CXL, future research should investigate the price-performance trade-off in multi-tier memory setups for slower and potentially cheaper CXL-attached memory.

### 3 Conclusion

With PMem, various assumptions about the homogeneity of DRAM access have been disrupted, leading to new challenges and designs. In this paper, we discussed how insights from these designs also apply to future CXL research. New CXL-based approaches need to focus on performance generalizability under initially limited hardware availability. They should consider how interaction with long-established components, such as prefetchers, impacts performance. And finally, in multi-tier memory setups, new designs should consider their economic viability as a key trade-off. While PMem is discontinued for now, we hope that future CXL work builds on these insights to establish a more general understanding of how systems interact with multi-tier memory.

**Acknowledgements:** This work was partially funded by the German Ministry for Education and Research (01IS18025A/01IS18037A), the German Research Foundation (414984028), and the European Union’s Horizon 2020 research and innovation programme (957407).

## References

- [BMR21] Benson, L.; Makait, H.; Rabl, T.: Viper: An Efficient Hybrid PMem-DRAM Key-Value Store. en, Proceedings of the VLDB Endowment 14/9, pp. 1544–1556, 2021.
- [BPR22] Benson, L.; Papke, L.; Rabl, T.: PerMA-bench: benchmarking persistent memory access. en, Proceedings of the VLDB Endowment 15/11, pp. 2463–2476, July 2022, ISSN: 2150-8097, URL: <https://dl.acm.org/doi/10.14778/3551793.3551807>, visited on: 11/22/2022.
- [Co22] Compute Express Link Consortium, I.: Compute Express Link (CXL) Specification, Revision 3.0, Version 1.0, <https://www.computeexpresslink.org/download-the-specification>, 2022.
- [Da21] Daase, B.; Bollmeier, L. J.; Benson, L.; Rabl, T.: Maximizing persistent memory bandwidth utilization for OLAP workloads. In: SIGMOD '21. tex.ids=daaseMaximizingPersistentMemory, ACM, 2021.
- [De22] Dell Technologies: Dell Rack Servers, <https://www.dell.com/de-de/workshop/deals/enterprise-deals/poweredge-rack-server-deals>, 2022.
- [GZ22] Gelsinger, P.; Zinsner, D.: Earnings Call Comments from CEO Pat Gelsinger and CFO Dave Zinsner, <https://www.intel.com/content/www/us/en/newsroom/news/intel-reports-first-quarter-2022-financial-results.html>, 2022.
- [Hw18] Hwang, D.; Kim, W.-H.; Won, Y.; Nam, B.: Endurable Transient Inconsistency in Byte-Addressable Persistent B+-Tree. In. Pp. 187–200, 2018, ISBN: 978-1-931971-42-3, URL: <https://www.usenix.org/conference/fast18/presentation/hwang>, visited on: 10/22/2021.
- [Lu20] Lu, B.; Hao, X.; Wang, T.; Lo, E.: Dash: scalable hashing on persistent memory. Proceedings of the VLDB Endowment 13/8, pp. 1147–1161, Apr. 2020, ISSN: 2150-8097, URL: <https://doi.org/10.14778/3389133.3389134>, visited on: 07/06/2020.
- [Re18] van Renen, A.; Leis, V.; Kemper, A.; Neumann, T.; Hashida, T.; Oe, K.; Doi, Y.; Harada, L.; Sato, M.: Managing Non-Volatile Memory in Database Systems. In: SIGMOD '18. SIGMOD '18, ACM, Houston, TX, USA, pp. 1541–1555, May 2018, ISBN: 978-1-4503-4703-7, URL: <https://doi.org/10.1145/3183713.3196897>, visited on: 07/06/2020.



# Improving GPU Matrix Multiplication by Leveraging Bit Level Granularity and Compression

Johannes Fett<sup>1</sup> Christian Schwarz<sup>1</sup> Urs Kober<sup>1</sup> Dirk Habich<sup>1</sup> Wolfgang Lehner<sup>1</sup>

**Abstract:** In this paper, we introduce BEAM as a novel approach to perform GPU based matrix multiplication on compressed elements. BEAM allows flexible handling of bit sizes for both input and output elements. First evaluations show promising speedups compared to an uncompressed state-of-the-art matrix multiplication algorithm provided by Nvidia.

**Keywords:** GPU; Matrix multiplication

## 1 Introduction

GPUs are becoming increasingly more popular for data analytics and compute workloads with increasing memory demands. GPUs share the constraint of having a significantly smaller memory capacity compared to CPUs with DRAM. One approach to mitigate this issue is to use compression. Our focus is to explore how to perform calculations on already compressed data.

In this work, we introduce BEAM (bitwise efficient matrix multiplication), a novel concept to directly compute on compressed elements in GPU memory. Instead of using native data types, we offer bit level granularity for unsigned integer based matrix multiplications. BEAM calculates directly on compressed data on a bit level granularity. Different problems that arise from bit level computation on GPU are discussed and strategies to deal with them introduced and evaluated. This paper focuses on unsigned integer based matrix multiplication to demonstrate that even in unfavourable compute bound use cases, compression can still be beneficial. In Section 2 preliminaries about compression on GPU and GPU architecture are introduced. Section 3 focuses on a detailed description of BEAM and strategies dealing with overflows and calculation of output bit sizes. Section 4 deals with related work and section 5 summarises the contribution.

---

<sup>1</sup> TU Dresden, Database Research Group, Nöthnitzer Str. 46, 01187 Dresden, Germany {johannes.fett|christian.schwarz5|urs.kober|dirk.habich|wolfgang.lehner}@tu-dresden.de

## 2 Preliminaries

### 2.1 GPU Architecture

A Nvidia GPU consists of a large number of arithmetical logical units called CUDA cores. Groups of 64 CUDA cores form a functional block called streaming multiprocessor. A streaming multi processor also shares register memory and shared memory across all its cores. A shared L2 Cache and VRAM (global memory) is accessible by all streaming multiprocessors through a shared memory bus system. The total amount of global memory is up to 80 GB for current GPU generations. Shared memory per streaming multi processor ranges from 48 KiB to 64 KiB depending on the GPU generation. The programming Model of a GPU is called single instruction multiple threads. A large number of threads is spawned to perform a computation (kernel). A group of threads is called a block. The total amount of threads is partitioned into a number of blocks. Each block is assigned to streaming multiprocessor. Most instructions are performed in groups of 32 threads at once, which is called a warp.

### 2.2 Compression on GPU

Typically, integer calculations work on an element level granularity. With BEAM we introduce the ability to calculate elements on a flexible bit level granularity. Specifically, we demonstrate a matrix multiplication on compressed elements. Different approaches to integer data compression have been covered by [Da17]. By combining different compression algorithms, an improved compression rate can be achieved. However, in our experiments we assume that all elements are compressed by zero suppression.

## 3 BEAM

BEAM allows flexible matrix multiplications by allowing elements on a bit level granularity instead of typical byte based data types. In case of using zero suppression, empty bits can be removed from Integer based data types. Test data is generated accordingly to conduct experiments on different bit sizes of elements ranging from 1 to 64 bit per element. The input bit size is static across the elements within a matrix to avoid the need for a prefix sum to access data elements. The supported data format is only unsigned Integers between 1 and 64 bit size.

### 3.1 Output Bit Strategies

The desired output bit size can vary depending on different strategies. If statistical information of a matrix is known, there might be a lower possible output bit size that fits all elements. If

| Strategy             | Description   | Acronym |
|----------------------|---|---------|
| Same as input        | output bits same as input bits  | sai     |
| Ceil to Power of Two | output bits is the smallest power of two greater or equal to the input bits | c2p2    |
| Max Value            | largest required output value, maximum 64 bit                               | maxv    |

Tab. 1: An overview of different strategies for calculating the output bit size.

| Strategy | Description  | Acronym |
|----------|--|---------|
| Overflow | default CUDA behavior: wrap around zero and cycle through the value range  | ovf     |
| Saturate | Stay at the largest possible value. This behavior mimics the common Sigmoid activation function from machine learning applications | sat     |

Tab. 2: An overview of different strategies for dealing with overflows

| Memory layout            | Description  |
|--------------------------|--|
| Canonical Layout (naive) | One matrix element uses 64 bits in memory.                                   |
| Tight types              | A bit level element is packed into a single next largest native datatype     |
| Padded Slabs (slabs)     | As many complete matrix elements as possible are placed into one 64 bit slab |
| Tight Packing (nogaps)   | Memory is viewed as a contiguous bitstream                                   |

Tab. 3: Memory representations for matrix multiplication.

there is no available information, defining a maximum needed bit size to avoid an overflow is a safer approach.

### 3.2 Overflow Behavior

In the case the calculated element in the output matrix causes an overflow, our approach offers two different strategies to deal with overflows. Saturate will pin the result at the max value of the value range in case of an overflow. This is realized by a builtin GPU intrinsic. Overflow is the default CUDA behavior that will lead to values cycling through the value range in case of an overflow.

### 3.3 Matrix Memory Representation

Because the input (and output) bit length of elements is not fixed to powers of two, using the native C integer types is inefficient. Therefore we experiment with different memory layouts to increase the performance. The matrices are stored row-major, without loss of generality.

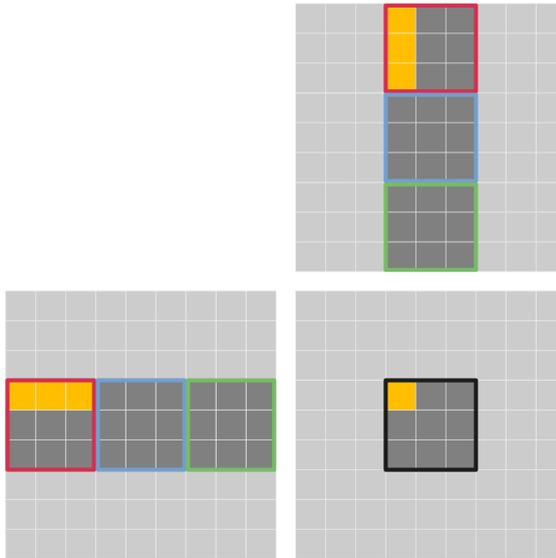


Fig. 1: Squarestride Matrix Multiplication example

For an overview of memory representations see Table 3. Slabs tries to fit as many bit level elements as possible into one 64 bit element. While this simplifies handling the memory it also leads to internal fragmentation. For example, 8 different 8 bit elements can fit without fragmentation into one 64 bit element. However, in case of 25 bits per element, only two will fit and consume 50 bits of space. The remaining 14 bits remain empty and lead to fragmentation. To avoid fragmentation the nogaps approach has been developed. In this case a contiguous bitstream is used to store all compressed elements. To allow simpler computations, end-of-row padding is introduced to the next 64 bit multiple.

### 3.4 Algorithms

This section contains an overview of all evaluated matrix multiplication approaches.

**Squarestride** Every thread block is responsible for exactly one sub-block of the output matrix and for every one of its threads for exactly one element within it (black outlined 3x3 box in the image 1). Both, this sub-block and the currently required sub-blocks for the left and right matrix, are kept in shared memory to reduce redundant reads from global memory. This sub-block based computation works such that sub-blocks are loaded one by one going inwards. Start by loading both red outlined sub-blocks and do partial computation, then continue to load both green sub-blocks. This pattern repeats until all results values have

been calculated. If the matrix is not a multiple of the sub-block dimensions, the excess values are assumed 0, such that they stay neutral to the computation. This avoids additional bounds checks.

**Flex out** Because access and computation logic calculation is in some cases significantly more complex if support for variable number of output bits (unequal to the input bits) is added, both version are provided for fair comparison. The more flexible version is suffixed with flex out.

**Baseline Guide** This kernel uses the squarestride strategy and the naive memory layout. This is the approach described in the Nvidia programming guide matrix multiplication example 2 [Nv].

**Tight Types** Tight types is using the squarestride strategy. Input bitsize is rounded up to the next native datatype. For example a 31 bit element would be packed into one 32 bit integer.

**Squarestride and Slabs** This kernel uses the squarestride strategy and the slabs memory layout. The slabs memory layout is used in the shared memory sub-blocks as well. In this case one thread no longer handles one element of the output matrix but one slab. Because that would make the sub-blocks rectangular by the increased number of vertical slabs required, each thread handles as many rows as there are elements in one slab.

**Squarestride Nogaps and Shared Memory Slabs** This kernel uses the squarestride strategy and the nogaps memory layout. It is very similiar to the squarestride kernel with the difference of using the noslabs memory layout in global memory. For faster computations, the slabs memory layout is still used in shared memory. This also avoids stitching together an element from two slabs during the computation within the individual sub-blocks.

**Squarestride Flex Out** This kernel is a version of matrix mul squarestride that supports a variable number of output bits. To achieve this, the number of elements stored in each shared memory slab is reduced to the number of elements in an output slabs. Note that the bit size within the shared memory is still only following the input bit count. Flex out means, that different output strategies are covered by the same kernel.

| GPU             | GPU Generation | CMake  | CUDA | NVCC     | G++      |
|-----------------|----------------|--------|------|----------|----------|
| RTX 8000 Quadro | Turing         | 3.25.1 | 11.5 | 11.5.119 | 9.3.0-17 |

Tab. 4: An overview of used build and compilation tools as used in the evaluation.

**Squarestride Nogaps Shared Memory Slabs Flex Out** This kernel is a version of matrix mul squarestride nogaps that supports a variable number of output bits. This is achieved using the same adaption as described by flex out. Slabs are being held in shared memory.

## 4 Evaluation

In this section, all of BEAMS algorithms are compared against the state-of-the-art Nvidia matrix multiplication. While all approaches receive the same input data, Nvidias approach works on an element level granularity with naive memory layout mentioned in 3. To allow a fair comparison, a variant of Nvidias matrix multiplication algorithm has been created that supports the saturated overflow behavior 2. As a rising trend, both general data sizes and machine learning models are growing in size. Thus, we evaluate both a small 64 MiB matrix multiplication and a larger 1 GiB matrix. Each data element is generated with varying bit size ranging from 1 to 64 bit. Bit sizes smaller than 64 bit use zero suppression to create a compressed data element. The Nvidia approach uses each element as 64 bit element. All experiments run with 3 repetitions. The average run time of all three is used in the evaluation. All experiments run on CUDA Cores and no tensor cores are used. Cublas is not used as frame of reference, as it does not support 64 bit Integer Operations. As BEAM is designed for Integer calculations, comparing against Cublas would be unfair.

### 4.1 Implementation

All experiments have been implemented in CUDA and C++. As build system Cmake is used. The Code builds with Clang and Nvcc. The experiments have been performed on an nvcc based binary.

### 4.2 Experimental setup

All experiments have been conducted on a Nvidia RTX 8000 GPU. For an overview of the system see Table 4. The GPU offers 48 GB GDDR6 Memory with a total theoretical maximum bandwidth of 672 GB/s.

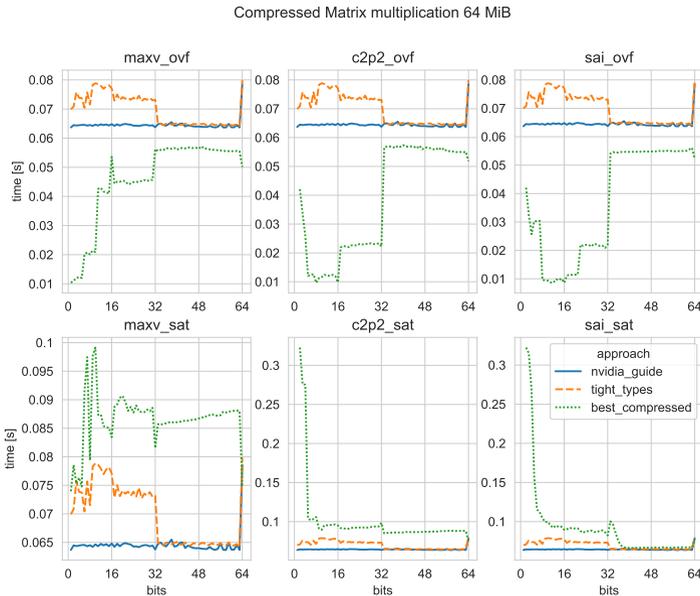


Fig. 2: 64 MiB Matrix multiplication. Compressed bitwise algorithms compared against Nvidia Matrix Guide and tight types approach. The best performing compressed algorithm is picked per point.

### 4.3 Results

Figure 2 shows an overview of different experiments on a 64 MiB data set. The grid of images is based on the chosen output bits approach in X direction as described in Table 1. The Y axis of the grid shows both different overflow approaches as shown in Table 2. Three different approaches are shown in each graph. Tight types and Nvidia guide have been explained in Section 3.4.. Best compressed is the best performing compressed algorithm per bit from a pool of all mentioned algorithms in Section 3.4.

In case of saturated overflow behavior, Nvidia’s Matrix algorithm shows the best performance. Tight types performs better than best compressed across all three different bit output strategies. For saturated overflow behavior on 64 MiB Matrices, Nvidia’s approach is the best. However, this changes if the overflow behavior is allowing overflows instead of saturating. In cases where the matrix multiplication will not overflow because the output data size is sufficiently large, the best compressed approach outperforms both, the Nvidia approach and tight types massively.

The size of data sets is constantly growing for machine learning and data management. To accommodate this trend, another experiment has been conducted on matrices with the size of 1 GiB. The experiment follows the same approach as the 64 MiB one. The results are

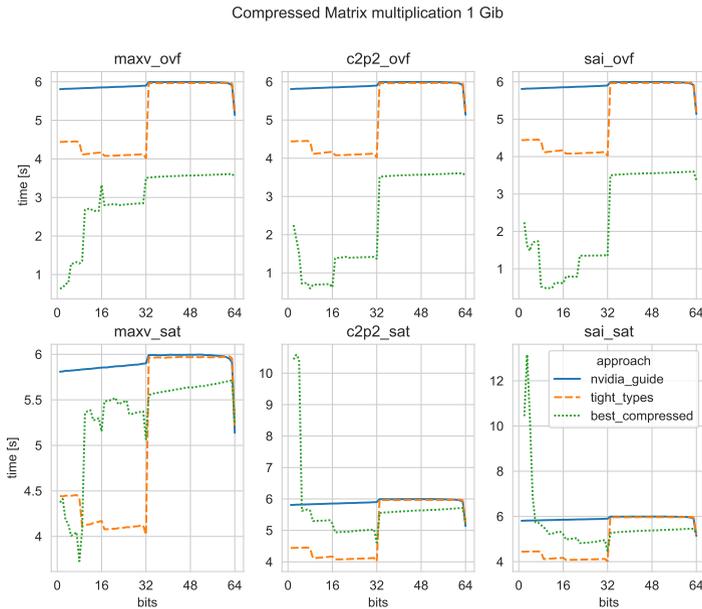


Fig. 3: 1 GiB Matrix multiplication. Compressed bitwise algorithms compared against Nvidia Matrix Guide and tight types approach

shown in Figure 3. In case of overflow behavior allows overflow, an increased speedup compared to the 64 MiB experiment can be achieved.

**Saturated 1 GiB** For saturated overflow behavior the result is vastly different and the Nvidia approach is the overall worst performing one. For the bit range of 33 to 64, tight types mirrors the Nvidia guide behavior, while best compressed outperforms both of them. The reason for this behavior is that tight types is only able to put one compressed element into one natively supported data type. Thus, a 33 bit unsigned integer leads to a 64 bit unsigned integer in case of tight types. Below 33 bits, tight types offers a speedup of about 1.4x compared to Nvidia’s approach. For both output bit behaviors ceil two power two and same as input combined with saturated overflow behavior, tight types is the best solution for elements ranging from 1 to 32 bit size. In case of maximum value output behavior, the compressed approach offers a small speedup between 1 and 7 bits input bit size. In case of 64 bit, no compression occurs, which leads to similar results for all three approaches. For very low bit sizes, *best compressed* performs increasingly worse due to being more compute intense on a rising number of elements.

**Overflow allowed 1 GiB** Changing the overflow behavior to allow overflows, reduces the compute intensity of the algorithm. Instead of using an intrinsic to check for overflow and handling it with branching within the kernel, this approach is default GPU behavior and

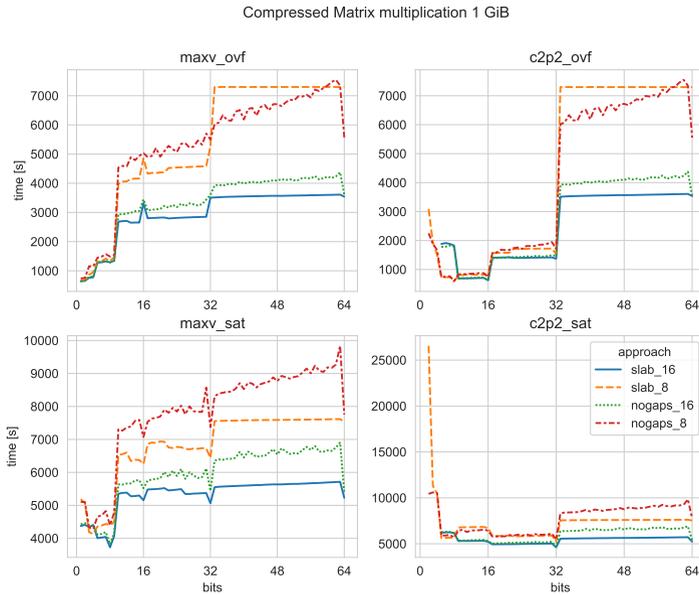


Fig. 4: 1 GiB Matrix multiplication. Comparison of different compressed computation algorithms

does not need any extra branches. Best compressed performs best across all input bit sizes with no exception. For larger output sizes in case of maximum value as chosen output bit strategy, best compressed performs slower than the other output bit strategies.

**Comparison of different compression approaches** Figure 4 shows 4 different compression algorithms, that are compared in detail. As the output bit strategy *same as input* is very similar to *ceil to power two*, it has been omitted from the graphs. Slabs is fitting into one element while ogaps features a contiguous bitstream. The missing data points indicate, that some CUDA configurations demand more shared memory than available. Overall the slabs approach outperforms nogaps with few exceptions. Within a block threads are 2 dimensional. Slab 16 for example uses 16 threads in two dimensions, which results in 256 total threads. Slab 8 only uses 8 threads per dimension, resulting in 64 total threads per block.

## 5 Related work

Shabag et. al have designed a compression framework integer based gpu computing [Sh22] called *tile-based lightweight integer compression*. It is based on storing compressed data in global memory and decompresses the data in shared memory. After computations, the data needs to be re-compressed and written back to global memory. The key difference to BEAM is, that BEAM does not need to decompress data before using it. However our

approach is limited to zero suppression, while Shabag et. al combine a number of different compression schemes to achieve higher compression rates. Also, their approach is focused on database operators and does not support matrix operations. Fang et. al have proposed compression for CPU GPU co-processing for databases [FHL10]. In this scenario, PCI-E becomes a major bottleneck, which can be improved by transferring compressed data.

## 6 Conclusion and summary

Our approach allows GPU matrix multiplication on a bit level granularity instead of the usual data element level granularity. It has been evaluated on different overflow strategies and output bit strategies. BEAM outperforms Nvidia slightly in case of a 64 MiB matrix and massively on a larger 1 GiB matrix. On average our approach offers a good speedup compared to the state-of-the-art approach. We look forward to further extend our bitwise GPU computation approach to other domains.

## References

- [Da17] Damme, P.; Habich, D.; Hildebrandt, J.; Lehner, W.: Insights into the comparative evaluation of lightweight data compression algorithms. Algorithms 1/1oranN, 1mappingdependingontheimplementation, 2017.
- [FHL10] Fang, W.; He, B.; Luo, Q.: Database compression on graphics processors. Proceedings of the VLDB Endowment 3/1-2, pp. 670–680, 2010.
- [Nv] Nvidia Matrix Multiplication Guide, <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#shared-memory>.
- [Sh22] Shanbhag, A.; Yogatama, B. W.; Yu, X.; Madden, S.: Tile-based Lightweight Integer Compression in GPU. In: Proceedings of the 2022 International Conference on Management of Data. Pp. 1390–1403, 2022.

# DNAContainer: An object-based storage architecture on DNA

Alex El-Shaikh,<sup>1</sup> Bernhard Seeger<sup>2</sup>

**Abstract:** The digital data volumes produced worldwide per year are ever-increasing. Estimates show that by 2025, we will have reached 175 zettabytes of globally created digital data. Despite today's advancements in storage devices, current database management systems cannot cope with these amounts of data. More than the recent improvements in storage technologies are needed to meet the ever-accelerating growth of generated data. This problem is further exaggerated when considering that current storage technologies such as HDD and tape require replacement every few years. To combat this deficiency, deoxyribonucleic acid (DNA) offers a novel durable (millennia scale), extremely dense, and energy-efficient storage medium. However, current DNA systems lack support for random access and more expressive query support beyond key-value lookups. In this paper, we present DNAContainer, a novel storage architecture on DNA that spans an ample virtual address space on objects, enabling random access to DNA at a large scale while adhering to required biochemical constraints. The interface of DNAContainer also facilitates the implementation of common external data structures, such as arrays and lists that store data in blocks of fixed size.

**Keywords:** DNAContainer; DNA Storage; Random Access; DNA Data Structures

## 1 Introduction

Current database management systems rely on solid state disks (SSDs), magnetic hard disk drives (HDDs), and tapes as their primary persistent memory devices [Bo16]. However, due to the dramatic increase in data produced daily, these devices will no longer cope with the amount of data soon. As stated in [Li20a], the increase in capacity of current data storage devices is already behind that of the data created. In addition, these traditional storage devices are rather expensive [Ma20] and require continuous replacement every few years due to their low durability [Bo16]. To address these severe problems, deoxyribonucleic acid (DNA) has recently been considered for managing persistent data. DNA is an extremely dense biomaterial holding up to 455 exabytes per gram, and thus at least six orders of magnitude denser than current devices [Bo16]. DNA endures several centuries and consumes around eight orders of magnitude less energy than traditional storage devices [Li20a, Zh16, A112]. Despite these apparent advantages, current technologies for reading and writing DNA induce a high latency (from hours to days). However, around 80% of generated digital

---

<sup>1</sup> University of Marburg, Department of Mathematics and Computer Science, Hans-Meerwein-Straße 6, D-35043 Marburg, Germany [elshaika@mathematik.uni-marburg.de](mailto:elshaika@mathematik.uni-marburg.de)

<sup>2</sup> University of Marburg, Department of Mathematics and Computer Science, Hans-Meerwein-Straße 6, D-35043 Marburg, Germany [seeger@mathematik.uni-marburg.de](mailto:seeger@mathematik.uni-marburg.de)

information worldwide is considered cold [Ap19, QSH22], i.e., the data is not accessed frequently, making DNA storage a potential candidate for the management of cold data. In addition, the cost of reading and writing DNA have declined dramatically over the past years [Li22].

Among the severe drawbacks to using DNA as storage are its unsatisfying direct access ability and its poor interface for reading and writing dedicated objects. Similar to blocks on traditional devices, DNA consists of oligonucleotides (oligos) that are contiguous subsequences, generally of fixed size. Furthermore, an oligo consists of a payload between a pair of DNA addresses, the so-called *primer pair*. The primers are addresses used for random access via Polymerase Chain Reaction (PCR) [Or18]. Due to strict biochemical restrictions on primers, only a few hundred primers exist in a DNA library, leading to a very small address space. However, there is a second approach to direct accessing oligos that uses microarrays with *barcodes* that are unique prefixes of the payloads. Then, the available address space grows up to several million [El22].

In this paper, we primarily address the second drawback that current DNA storage systems do not provide a coherent interface to write and read random data objects. The reason is that DNA does not offer a natural linear address space as it is known from disks and tapes. Instead, there is only a key-value approach that maps a data object identifier (DOI) directly to a barcode or primer. To access a requested data object, its DOI has to be translated into the DNA address before performing the actual read operation. A so-called *routing table* can manage the mappings on a traditional storage device. Furthermore, an insertion of a new object first generates a new barcode (or primer) and updates the routing table before storing the actual object on DNA. This direct approach of current DNA storage systems to accessing data on DNA has serious drawbacks. For example, the generation of barcodes is non-trivial because they have to be sufficiently different from the others.

In this paper, we introduce DNAContainer, a novel DNA storage architecture that offers a virtual address space of objects on DNA, including *put* and *get* operations, addresses translation, and rerouting of invalid addresses. Our system reads objects from DNA via microarrays that allow using the ample available address space. As a special case, DNAContainer also offers the same interface as block storage when the objects are defined as fixed-sized blocks. Furthermore, a block storage facilitates the direct implementation of essential external data structures like arrays and lists on DNA, hiding the actual complexity of a DNA device. For example, DNAContainer internally checks if the generated addresses are too similar by utilizing locality-sensitive hashing (LSH) and approximating the Jaccard similarity of two DNA sequences [IM98, Br97, Bu01, Be15]. In addition, DNAContainer supports error correction mechanisms such as Reed Solomon [RS60] to address the inherent problem of error-prone reading from and writing to DNA. Users of DNAContainer do not have to deal with these problems anymore and instead use the common interface of storage systems as it is known from other devices.

The remainder of the paper is structured as follows. The following Sect. 2 discusses recent works and studies on DNA systems and virtual address spaces. Section 3 introduces notation and terminology commonly used in the context of DNA storage. Section 4 provides the design and implementation of DNAContainer and its components. It further shows how to generate DNA addresses and payloads that adhere to certain biochemical constraints. In Sect. 5, we detail the implementation of basic data structures like array and list on DNAContainer. Section 6 presents experimental results of a simulation with DNAContainer managing millions of oligos. Finally, Sect. 7 concludes the paper.

## 2 Related Work

In the following we first discuss related work on DNA storage systems. Thereafter, we put our focus on approaches with virtual address spaces.

The approach in [Ap19] encodes relational data objects (records) interleaved with meta-information as oligos. The meta information contains, e.g., the table name of the record and its primary key. Reading oligos is achieved by utilizing primers and PCR. Since only a few primers are available due to the biochemical restrictions, i.e., the address space is small, the same primer is used to address multiple records. For example, to read a specific record from DNA, a pre-known primer is used to fetch all oligos tagged with that primer. The encoded meta-information is further used to return the desired record, e.g., by its primary key. Moreover, the meta-information encoded within each oligo significantly decreases the information density per oligo, and the realized storage capacity is around  $\approx 16.5\%$ .

In [Or18], 35 different files were placed in a separate DNA pool each, resulting in 200MB of information. Since PCR utilized random access, this physical separation of files was necessary to overcome the imposed restrictions on the limited available primers. Additionally, there are 35 physical addresses, each of which resembles a physical location of a single tube with one file, which significantly decreases information density over all tubes.

Fountain codes were used in [EZ17] to encode 2.15MB of data plus 7% redundancy. Similar to our previous work [EI22], fountain codes provide a direct way to tune redundancy and are very practical for DNA encoding. Nevertheless, the work in [EZ17] utilizes PCR for retrieving data and does not support random access at a large scale.

So far, PCR is still the standard technology to read data from a DNA pool. In [Li20b], an alternative technology called *DORIS* is proposed to overcome PCR limitations yielding a larger address space at around 12,000 available addresses. However, even 12,000 addresses are not sufficient to exploit the massive storage capacity of DNA.

The random access approach presented in [Ba20] encodes data physically encapsulated in impervious silica capsules that are surface-labeled with selected DNA sequences called barcodes. These barcode labels re-emit light when excited. Hence, each file is labeled with specific barcodes and is detected by special optical channels. For example, the file "bird" can

be detected with the barcode “can fly” and so on. However, only labeled files can be detected. Additionally, special equipment, such as optical channels, is needed.

According to [CNS19, Xu21], most recent studies do not support random access to their DNA storage system. These systems require a 5 to 3,000-fold physical and logical redundancy to reduce errors, substantially reducing storage density. In addition, many DNA systems fail to encode information such that the resulting DNA is sufficiently stable for long-time archival. In particular, many DNA systems fail to restore the original data objects after reading the DNA [Wa19]. Furthermore, we are unaware of a system with virtual address space to access a data object. Instead, a user has to provide a primer for reading a data object. These primers must be managed on a traditional storage device. More complex queries beyond simple key-value queries are not supported on data collections. In particular, data structures like lists and arrays are not supported in any system, making data management difficult. Moreover, we use barcodes to exploit the large available address space [El22], whereas most current systems still rely on PCR and primers, and thus only support a small address space.

There is a plethora of work related to virtual address spaces in computer systems. For example, a few object-oriented database systems like O2 [De90] have used an address transformation table to convert unique object addresses visible to the user into internal addresses. In addition, a flash disk also offers a similar mapping known as the flash translation layer (FTL) to implement wear leveling [MFL14]. However, the designs of these approaches do not consider the unique features of DNA storage and thus are not directly applicable.

A common problem of today’s storage technologies is successfully restoring archived data after several decades of writing the data to storage devices [AJ20]. For example, as mentioned above, current storage technologies such as flash memory rely on FTL, which requires storing meta-information about the corrupted memory cells to keep the device functioning. Current DNA systems manage the used primers (or barcodes) on a traditional storage device and face a similar problem. That is, if the used primers and barcodes are lost, the data on DNA cannot be restored. However, for DNA, storing the required meta-information also on DNA might solve this problem. DNA is an omnipresent material, and its principal building structure has never changed over millions of years and is expected to be ubiquitous for millions of years in the future.

### **3 Preliminaries**

DNA is a long molecule found in all known living organisms. It carries the genetic code, such as instructions, functions, and reproduction in living organisms, including some viruses. Moreover, DNA is composed of smaller units called nucleotides. A nucleotide contains one of the following nucleobases: Adenine (A), Thymine (T), Cytosine (C), or Guanine (G). These nucleotides’ specific combination and order make up living organisms’ different instructions and functions. Finally, DNA is composed of two polynucleotide strands of the same length that loop and twist around each other to form a double-helix. Each nucleotide of one chain

pairs and forms hydrogen bonds with the corresponding nucleotide from the other strand. According to the canonical Watson-Crick pairing [Sp59], A binds to T and G to C. Hence, we say A is complementary to T, C is complementary to G, and vice versa.

In the following subsections, we will introduce key terms, such as *DNA pool* and *hybridization*, typically used in the DNA storage context.

### 3.1 DNA Pool and Library

A DNA pool is a collection of one or more double-stranded DNA fragments held *in-vitro*, i.e., outside of a living organism, in a single container or test tube. Typically, one container refers to a single pool, whereas multiple pools represent a library. Nevertheless, a single pool can also be referred to as a library.

### 3.2 Encoding and Decoding

The term encoding is used to describe the process of transforming binary data to DNA, i.e., instead of bits, the information is represented by nucleotides. On the other hand, decoding describes the transformation of nucleotides back to the *original* binary representation.

### 3.3 Denaturation and Hybridization

Double-stranded DNA is generally stable under physiological conditions, meaning the bonds forming the double-helix will remain bonded [Ch99, YPFK06]. However, as illustrated in Fig. 1, raising the surrounding temperature, e.g., in a laboratory, will cause the strands to separate as single-stranded DNA (ss-DNA). This process is called denaturation. Therefore, lowering the temperature will allow the ss-DNA to bind together as double-stranded DNA (ds-DNA), which is called hybridization.

### 3.4 DNA Synthesizing and Sequencing

DNA synthesis is writing DNA by linking and joining nucleotides together, forming a single-stranded sequence. Today's technologies allow near-perfect DNA synthesis for over thousands of DNA fragments in parallel. However, a small error can already lead to a significant decrease in product quality and redundancy is introduced to avoid these errors. Thus, modern sequencing machines [KC14] read the same sequence multiple times. Both synthesizing and sequencing costs have been declining dramatically over the past years, and sequencing productivity has already outpaced Moore's law by 2008 [Ap19]. However, sequencing machines are designed for reading an entire DNA and not for random access so far.

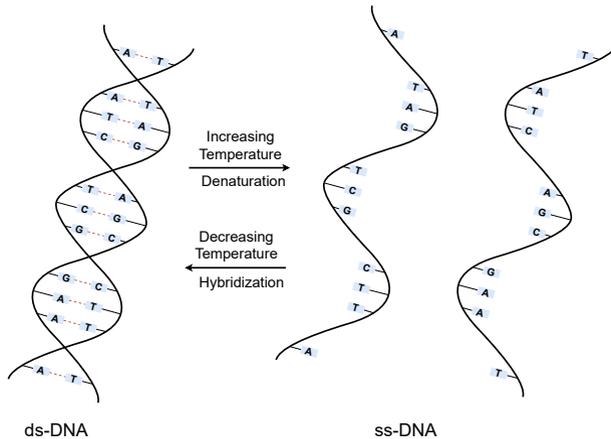


Fig. 1: DNA denaturation and hybridization.

### 3.5 Reading DNA with a Microarray

A microarray consists of a small surface (in the size of today's HDD) usually made of glass. It contains DNA *sites* to which DNA sequences can be immobilized or printed [Ku01]. The array can fetch a DNA sequence by printing its complementary sequence to one of its sites. These printed DNA sequences are often called *barcodes* or *probes*, but we simply call them DNA addresses. For example, a microarray with 20 printed DNA addresses can simultaneously fetch 20 oligos from an oligo pool. This procedure is done as follows. First, the oligo pool's temperature is raised, denaturing the contained DNA. Next, the denatured oligos are placed onto the microarray. Then, the temperature is lowered to allow the single-stranded oligos to hybridize to their complementary sites on the array. Finally, the array is washed, removing all the remaining oligos that did not hybridize, i.e., bind to any of the array's sites. The obtained bonded oligos are sequenced, and the data is transferred to a computer for further analysis. Note that a microarray can fetch a sequence  $s$ , even if only a complementary subsequence of  $s$  is printed to the array. Today's microarrays can contain up to several million sites [Bu13], allowing access to millions of DNA sequences simultaneously.

### 3.6 DNA Constraints

As described above, sequencing and synthesizing DNA is error-prone. For example, it is well-known that DNA sequences with a too high or low number of G's and C's causes a high error probability in the sequencing process [Sc20]. Hence, to reduce errors, our generated DNA codes must adhere to the following constraints:

1. The number of G's and C's (*GC content*) should be around 50%.
2. Consecutive repeats of the same nucleotide (*Homopolymer*) should be avoided.
3. Mutual overlaps of DNA addresses should be avoided.
4. Mutual overlaps of the oligos should be avoided.

The first and second constraints considerably reduce sequencing and synthesizing errors [Sc20]. Constraint (3) ensures that the microarray treats every DNA address uniquely. Finally, constraint (4) guarantees that a DNA oligo does not carry a DNA address as a payload.

## 4 The Design of DNAContainer

This section describes the architecture and functionality of DNAContainer. DNAContainer provides an interface for writing binary data to and reading it back from DNA into the memory of a computer system. It manages a DNA pool consisting of oligos of the same length  $L_{oligo}$ , similar to a block on common storage devices. Each oligo is composed of an address and a payload. Addresses are of the same length  $L_{address}$ , and payloads are then of length  $L_{payload} = L_{oligo} - L_{address}$ . Current DNA synthesis and sequencing costs are typically lower for shorter oligos ( $L_{oligo} \leq 250$ ) than for longer ones [HMG19, GMM16]. Thus, the size of an oligo is substantially smaller than a typical block size. Figure 2 provides an example of an oligo of  $L_{oligo} = 18$ ,  $L_{address} = 6$ , and  $L_{payload} = 12$ .

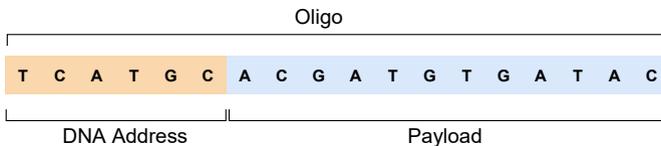


Fig. 2: The composition of an oligo in DNAContainer.

Suppose a large data object like a block has to be written to DNA, exceeding the size of an oligo. In that case, DNAContainer splits the data object into multiple segments, each of which fits into an oligo's payload. To read the data object back from DNA, DNAContainer first computes all DNA addresses of the relevant oligos. Then, a microarray retrieves the corresponding oligos, and finally, the oligos are assembled and decoded such that the object (block) is in memory again.

In the following, we give an overview of the functionality of DNAContainer, which can manage a set of objects in a linear address space. If objects refer to fixed-size blocks, DNAContainer offers the standard interface of block-based storage. In contrast to traditional devices, however, objects are not required to be of the same length. Rather than using block, we prefer using the generic term objects instead.

Each data object written to the DNA storage is tagged with a unique integer number *Id* obtained from a linear virtual address space. Furthermore, the *Id* is translated to a DNA address and vice-versa (see Sect. 4.1), creating an unambiguous mapping  $Id \leftrightarrow \text{DNA address}$ . The *Id* is a virtual address visible to the user, while the associated DNA address refers to the root oligo of the object. In particular, a user can read the associated data object from DNA by simply using the virtual address. Similar to bad blocks on disks, this mapping ensures that virtual addresses are usable, which is not valid for the underlying DNA addresses. This process is further explained in Sect. 4.1.2. Furthermore, the data object, i.e.,

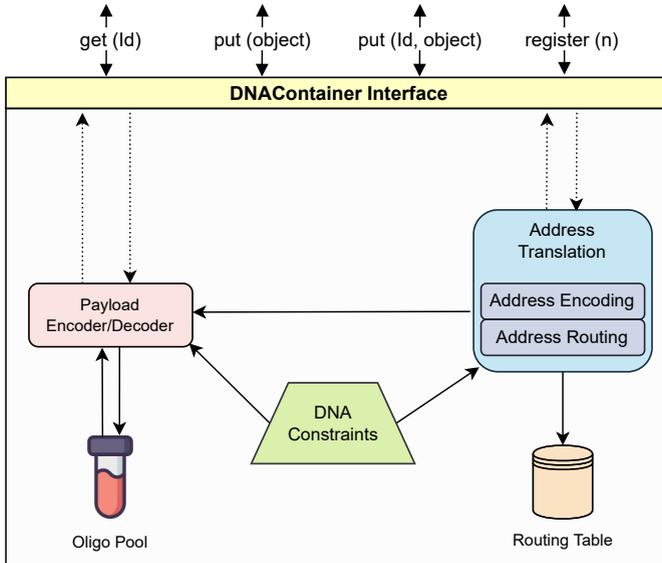


Fig. 3: Overview of DNAContainer and its components.

the information in an oligo's payload, can be encoded with different methods that we mention in more detail in Sect. 4.2. Even if the payload's encoder returns a payload that does not adhere to the constraints in Sect. 3.6, DNAContainer implements additional optimizations such that it fulfills the required constraints, which are discussed in Sect. 4.3. Figure 3 provides an overview of DNAContainer and its components. To sum up, DNAContainer is composed of the following main components: address translation to map an *Id* to a DNA address and vice-versa, address routing to map a DNA address to a new valid DNA address, the payload encoder, the payload decoder, and the DNA pool where the data is stored.

Our interface provides an abstraction layer to the methods mentioned above. In particular, to write a data object to DNA, we use the function `put`, the function `get` to read a data object from DNA, and `register` to pre-register an *Id* that can be used to write a data object at some point in the future, using that *Id*. The following discusses each of these functions in more detail.

**register.** As mentioned above, our DNA system assigns each data object a unique Id. The function `register` returns a unique Id that is not assigned a data object yet. In other words, this Id is reserved for a future data object that can be stored later. Note that the returned Id is not yet mapped to a DNA address and is only done once an actual data object is to be stored in the DNA storage. We implement the function `register(n)` that returns  $n$  newly registered consecutive Id numbers.

**put.** This function represents the write operation on DNA. It is used to store a data object in the DNA storage. We implement two variants of `put`. The first variant takes a data object, stores it in the DNA storage, and returns a newly registered unique Id. This newly registered Id is calculated by calling `register(1)`. The second variant takes an Id that was previously registered along with a data object and stores this data object in the DNA storage given the Id. We can extract the data object from the DNA storage for both variants by calling `get(Id)`. The call `put(Id, d)` can be used to replace the currently stored data object at Id with  $d$ . This is done by rerouting the virtual address Id (see Sect. 4.1.2), and the oligos of the old data object are not physically removed by default. However, to physically remove a data object from DNAContainer, the corresponding oligos can be fetched using `get` and discarded. Hence, the used addresses of physically removed objects can be reused.

**get.** This operation represents the read operation from DNA. By providing an Id to the function, `get(Id)` returns the data object associated with that Id. Hence, `get` is the inverse of `put`. For example, the following equality  $d = \text{get}(\text{put}(d))$  holds for any data object  $d$ .

## 4.1 Address translation

DNAContainer provides its interface based on the virtual address space on integers. The `put` operation writes a data object into the DNA storage by generating a new Id, which is translated to a DNA address. The data object is encoded to the payload, and the oligo is formed by annealing the DNA address and the obtained payload. The following sections detail the encoding of data objects as payloads and the translation of Ids to DNA addresses.

### 4.1.1 Address Encoding

We utilize the method described in [Go13] to encode an Id to a DNA address. First, the Id is converted to a string of bytes by mapping every digit in base 10 to a byte character. Next, the string is compressed with a static Huffman code of base three. Then, each of the obtained Huffman digits is mapped to a nucleotide, forming a DNA sequence. The obtained DNA sequence could be longer than  $L_{address}$ ; thus, we set  $L_{address}$  to be sufficiently large.

Note that this method is reversible, i.e., following each mentioned step backward leads to the Id used.

Furthermore, the obtained DNA sequence could be shorter than  $L_{address}$  or even violate the constraints mentioned in Sect. 3.6. In that case, we apply the optimizations explained in Sect. 4.3. The optimizations always return a DNA address of length  $L_{address}$ . However, if the sequence after optimizations does not adhere to the required constraints, we route the used Id to a new Id, which is explained in the following section.

### 4.1.2 Address Routing

Suppose a DNA address obtained after encoding and optimizations fails to fulfill the given constraints in Sect. 3.6. In that case, the used Id is mapped (routed) to a new Id as shown in Algorithm 1. Let us refer to the Id as  $Id$ , the new Id as  $Id^R$ , and the mapping  $Id \mapsto Id^R$  as the routing table. As depicted in Fig. 3, the address translation manages the routing table  $Id \mapsto Id^R$ , which is stored on a traditional storage device. The routing table must be read before accessing the DNA storage. The new  $Id^R$  is encoded with the same method mentioned in the section above.

---

#### Algorithm 1: Routing $Id$ to $Id^R$

---

**Input:**  $Id$

**Output:**  $Id^R$

```

1  $Id^R := Id$ 
2  $addr := \text{asDnaAddress}(Id^R)$ 
3 while not  $\text{constraints.adhere}(addr)$  do
4    $Id^R := Id^R + 1$ 
5    $addr := \text{asDnaAddress}(Id^R)$ 
6  $\text{routingTable.put}(Id \mapsto Id^R)$ 
7 return  $Id^R$ 

```

---

As shown in Algorithm 1, the routing finishes once a new  $Id^R$  that fulfills the constraints is found. Note that  $Id$  and  $Id^R$  could be equal if the Id already adheres to the constraints. The algorithm iterates over the integers  $Id^R = Id, Id + 1, Id + 2, \dots$ , translating each to a DNA address, only stopping once it finds  $Id^R$  of which the DNA address fulfills all the required constraints.

## 4.2 Payload Encoding

There are several approaches to how to encode a data object as DNA. We refer to the data object as a stream of bytes, which can be mapped to DNA nucleotides. For example, a straightforward method is to map every two consecutive bits to a respective DNA nucleotide,

e.g., 00  $\mapsto$  A, 01  $\mapsto$  C, 10  $\mapsto$  T, and 11  $\mapsto$  G. In that case, a data object consisting of long runs of zeros or ones would get mapped to homopolymers, violating the required constraints in Sect. 3.6. More sophisticated methods [Go13, EZ17, Do20, EI22] have been proposed, providing DNA codes that adhere to some or all the required constraints regardless of the input byte stream. For DNAContainer, any method encoding the data object to DNA nucleotides can be used because we apply optimizations (see Sect. 4.3) to return payloads that adhere to all of the mentioned constraints. However, by utilizing a fountain code [EZ17] to encode the payload, we already obtain DNA codes that obey the constraints (1) and (2) and include error correction, which leaves optimizing for the remaining constraint (4).

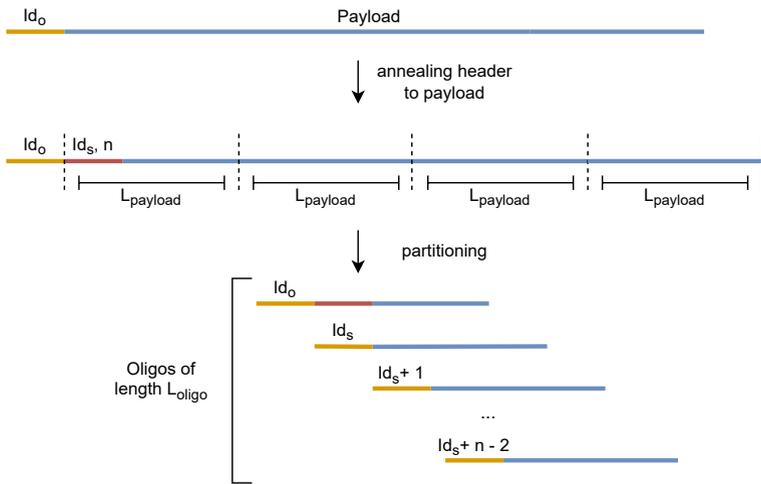


Fig. 4: Partitioning long oligos in DNAContainer.

Furthermore, if the given data object is too large, meaning that the payload is longer than  $L_{payload}$ , then the payload is partitioned among multiple oligos. This procedure is illustrated in Fig. 4 and is explained in the following. Let us assume that  $Id_o$  is the registered Id used to store the data object. First, we need to calculate the number of oligos  $n$  to which the long payload is split. Next, we register  $n - 1$  consecutive Ids, referred to as  $Id_s + i$  for  $i = 0, \dots, n - 2$ . Then, we anneal  $Id_s$  and  $n$  to the left end of the long payload. We will refer to these two numbers as the payload's *header*, marked in red in Fig. 4. Note that we map the integers to DNA by representing them in base four and finally map each base four digit to a corresponding nucleotide. Therefore, the occupied space of the payload's header is always the same. After that, we split the payload (payload plus header) into partitions  $p_i, i = 0, \dots, n - 1$  of the size  $L_{payload}$ . Finally, we need to address the obtained partitions. The first partition  $p_0$  is addressed by  $Id_o$ , and each following partition  $p_i, 1 \leq i \leq n - 1$  is addressed with  $Id_s + i - 1$  as shown in Fig. 4. This method only works if the length of the payload plus its header are divisible by  $L_{payload}$ . In the other case, the last split partition  $p_{n-1}$  would be smaller than  $L_{payload}$ . To solve this issue, we add specific padding

to  $p_{n-1}$ , resulting in all obtained oligos having the same length. Note that the function `put` implements the partitioning procedure and returns  $Id_o$  for the given data object.

To further optimize the oligos (after partitioning), we initially split a long payload into smaller payloads, each shorter than  $L_{payload}$ , and further add specific padding to each of them. This added padding is used to adjust the GC content and is further explained in the following section.

To decode the obtained oligos, i.e., to read the encoded data object, we read the first oligo given by  $Id_o$ . Next, we decode  $Id_s$  and  $n$  from the obtained payload to obtain the next virtual addresses  $Id_s + i$ ,  $i = 0, \dots, n - 2$ . Finally, we read the corresponding oligos, assemble the payloads, and decode the data object.

### 4.3 Optimizing DNA Sequences

This section details the optimizations applied for DNA addresses and payloads. We implement two optimization steps. The first adds specific padding to a given DNA sequence, correcting its GC content closer to 50%. The second generates a number of permutations of the DNA sequence, selecting the one that adheres to all the constraints in Sect. 3.6.

#### 4.3.1 Padding

If a DNA sequence is too short and the GC content is not 50%, we append additional nucleotides until the desired length is reached. For padding, we use 11 pre-computed DNA sequences  $s_i$ ,  $i = 0, \dots, 10$  where  $s_i$  has a GC content of  $\frac{i}{10}$ . To add padding to a given DNA sequence  $a$ , we append nucleotides from the padding sequence  $s_i$  that best corrects the GC content of  $a$  to 50%. To mark the position at which the padding starts, we first append a specific delimiter sequence that is not contained as a subsequence in any  $s_i$ . Let us illustrate this with an example. Suppose  $a = (\text{TCATT})$  with a GC content of 20%, and the target length of  $a$  is  $t = 12$ . Let the delimiter be  $d = (\text{GT})$ . The sequence  $a$  after appending the delimiter sequence is  $a_d = (\text{TCATTGT})$  with a GC content of  $\approx 28.5\%$ . There are 5 remaining nucleotides to add from one of the pre-computed padding sequences. Since  $a_d$  has a GC content lower than 50%, we need to add padding information with a GC content higher than 50% to obtain a sequence with an overall GC content of nearly 50%. We evaluate the index  $i$  of the ideal padding sequence  $s_i$  as:

$$i = \left\lfloor 10 \cdot \max \left\{ 0, \frac{\frac{t}{2} - |q|_{\{G,C\}}}{t - |q|} \right\} \right\rfloor \quad (1)$$

The expression  $|q|$  returns the number of nucleotides in  $q$ , and  $|q|_{\{G,C\}}$  returns the number of nucleotides in  $q$  that are either a G or a C. Plugging in  $t = 12$  and  $q = a_d$  in Eq. (1),

we obtain the padding sequence's index  $i = 7$ . Let us assume  $p_7$  was pre-computed as  $p_7 = (\text{TGCGGCTCCA})$ . Hence, to reach  $t = 12$ , we append the first 5 nucleotides from  $p_7$  to  $s_d$  resulting in  $(\text{TCATTGTTGCGG})$  with a GC content of 50%.

The obtained DNA sequence after padding is likely to fulfill the constraint (1) in Sect. 3.6 but could still violate the remaining constraints. To adhere to all constraints, we further optimize the sequence after padding by permutations, which is explained in the following section.

### 4.3.2 Permutation

The DNA sequence obtained after padding could still contain homopolymers and have mutual overlaps with, e.g., other DNA addresses or payloads. To fulfill the remaining constraints (2), (3), and (4), we generate  $m$  permutations of the given DNA sequence, selecting the permutation that fulfills the constraints. We use the classical Fisher-Yates method [Du64] to compute a permutation. This method generates  $k - 1$  index pairs to be swapped, where  $k$  is the length of the sequence. This method requires sampling random numbers from a random numbers generator (RNG) that requires a *seed* for initialization. Two RNG instances initialized with the same seed produce the same random numbers in the same order. To calculate the seed of a DNA sequence  $q$ , we evaluate:

$$\text{seed}(q) = |q|_{\{A\}} \cdot |q|_{\{C\}} \cdot |q|_{\{T\}} \cdot |q|_{\{G\}} \quad (2)$$

where  $|q|_{\{b\}}$  counts the number of  $b \in \{A, C, T, G\}$  in  $q$ . Note that  $\text{seed}(q)$  is invariant of permutation, i.e., the seed of  $q$  and all its permutations is the same. Hence, we can reverse a permuted sequence to its original by knowing the seed. Finally, to compute the  $m$  permutations of  $q$ , we initialize  $m$  RNGs with  $\text{seed}(q) + i$ ,  $0 \leq i < m$  that are used to permute  $q$ . Additionally, we append the offset  $i$  to each permuted sequence. Therefore, to reverse a permuted sequence, first, we decode and remove the encoded offset  $i$  from the DNA sequence to obtain the permuted DNA sequence  $\hat{q}$  without offset. Next, we initialize an RNG with  $\text{seed}(\hat{q}) + i$ . Finally, using this initialized RNG, we swap the  $k - 1$  generated index pairs in reverse, i.e., starting from the  $(k - 1)$ -th index pair to the first index pair.

After permutation, the obtained permuted sequence has its GC content corrected and is not likely to contain any homopolymers. Constraints (3) and (4) are further checked by approximating the Jaccard distance between two DNA sequences using LSH according to [El22]. The Jaccard distance is a metric between 0 and 1. A Jaccard distance of 0 means that the given two DNA sequences are as similar as possible, and a Jaccard distance of 1 is the maximum dissimilarity two DNA sequences can have. Hence, we select the permuted sequence that simultaneously contains no homopolymers and maximizes the Jaccard distance to the other DNA sequences.

This optimization is applied to both DNA addresses and payloads. If a DNA address after padding and permutation still does not fulfill all constraints (1), (2), (3), and (4), we route its

corresponding  $Id$  to a new  $Id^R$  as detailed above in Sect. 4.1.2. If a payload does not fulfill the constraints, even after optimizations, we have the following options: (i) increase the number of permutations  $m$ , and (ii) incorporate the constraints into the payload's encoder and do not apply any optimizations to payloads. The first option to increase the number of permutations  $m$  is straightforward. More permutations increase the probability of finding a permuted DNA sequence that adheres to the constraints. The second option shifts the problem of generating payloads that adhere to certain constraints to the payload's encoder and is illustrated to work by utilizing fountain codes in [El22].

## 5 Implementing Data Structures on DNAContainer

Data structures provide useful abstractions and are necessary for efficient data access [Co22]. It is the basis for implementing efficient algorithms and even allows the integration of index structures directly on DNA. Hence, we implement three basic data structures: (i) Reference, (ii) Array, and (iii) List on DNAContainer, showcasing its usability.

### 5.1 Reference

This data structure (or data type) is implemented by the function `put`. In particular, by calling `put( $d$ )`, the data object  $d$  is written to DNAContainer, returning a unique  $Id$ . We call this  $Id$  the *reference* to  $d$ . Therefore, essentially, every data object in DNAContainer is stored by a reference.

### 5.2 Array

Arrays are a well-known construct that current programming languages implement and data management algorithms rely on. We implement the array construct on DNAContainer, enabling concurrent access to its elements using only one  $Id$ . Let  $Id_o$  refer to an  $m$ -elements array. We further assume that every element of this array is encoded to DNA, e.g., by a fountain code. To write this array to DNAContainer, we generate  $m$  consecutive  $Ids$  by calling `register( $m$ )`. Each of these  $Ids$  is used as a reference to an array's element. Let us refer to these  $Ids$  as  $Id_a, Id_a + 1, \dots, Id_a + m - 1$ , and to the  $i$ -th element of the array as  $e_i$  where the first element is  $e_0$  and the last is  $e_{m-1}$ . As illustrated in Fig. 5, we utilize the `put` function with which we store each array's element calling `put( $Id_a + i, e_i$ )`. Note that by calling `put`, the element could be partitioned to multiple oligos as explained in Sect. 4.2. Furthermore,  $Id_o$  is used to store  $Id_a$  and  $m$  as payload information.

To read an element of the array, we first call `get( $Id_o$ )` to obtain the payload encoding  $Id_a$  and  $m$ . After that, we can access any index  $i$  of the array by calling `get( $Id_a + i$ )`,  $i = 0, \dots, m - 1$ , returning the element  $e_i$  of the array. Moreover, we can read the entire array in parallel by calling `get` on each index of the array simultaneously.

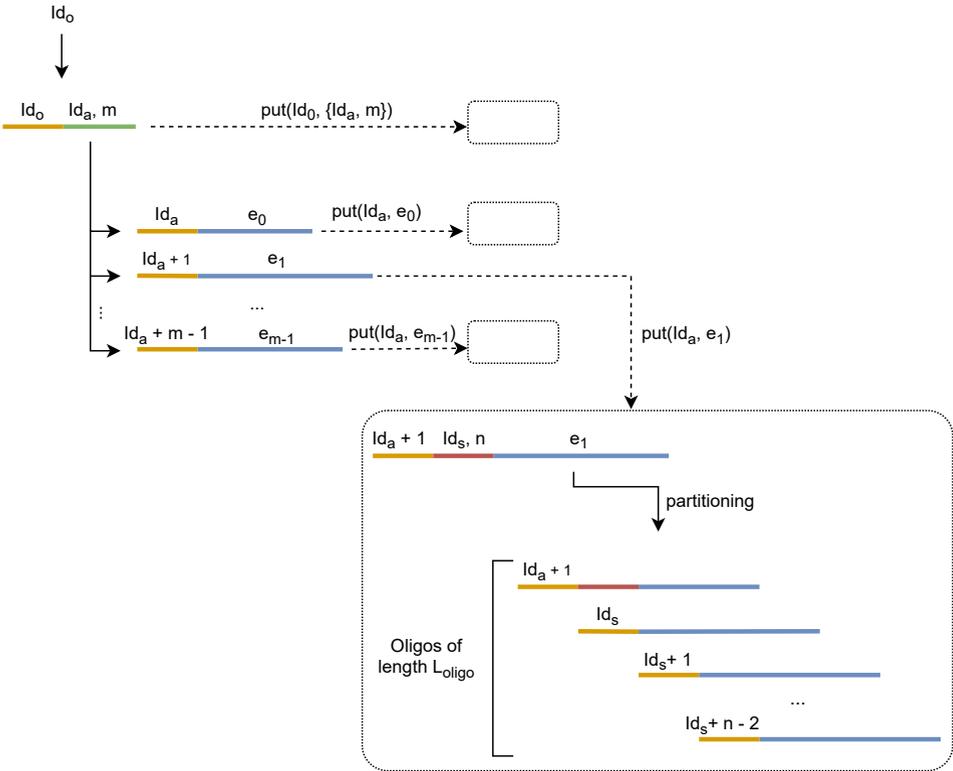


Fig. 5: Implementation of the array construct on DNAContainer.

### 5.3 List

Like an array, a list is a collection of elements where we assume the elements are mapped to DNA. However, unlike arrays, lists do not have a fixed size. DNAContainer implements a list as a chain of elements where each element points to the next one. Let us illustrate the implementation on DNAContainer by the example given in Fig. 6.  $Id_o$  is the Id used to reference the list of elements  $e_0$  and  $e_1$  marked as blue. The first element  $e_0$  is stored in DNAContainer along with a newly registered  $Id_1$  by calling  $put(Id_o, \{Id_1, e_0\})$  where  $Id_1$  is used to reference the next element of the list. Hence, element  $e_1$  is stored in DNAContainer along with the next newly registered  $Id_2$  and is referenced by  $Id_1$  by  $put(Id_1, \{Id_2, e_1\})$ . Since we invoke a put operation every time we append an element to the list, each element could be partitioned as detailed in Sect. 4.2. We repeat this procedure for every element appended to the list. For example, adding a third element  $e_2$  to the list is done by storing  $e_2$  along with a newly registered  $Id_3$  and referenced by  $Id_2$ . This is a crucial difference to an array, where the array structure does not support adding elements after referencing the array.

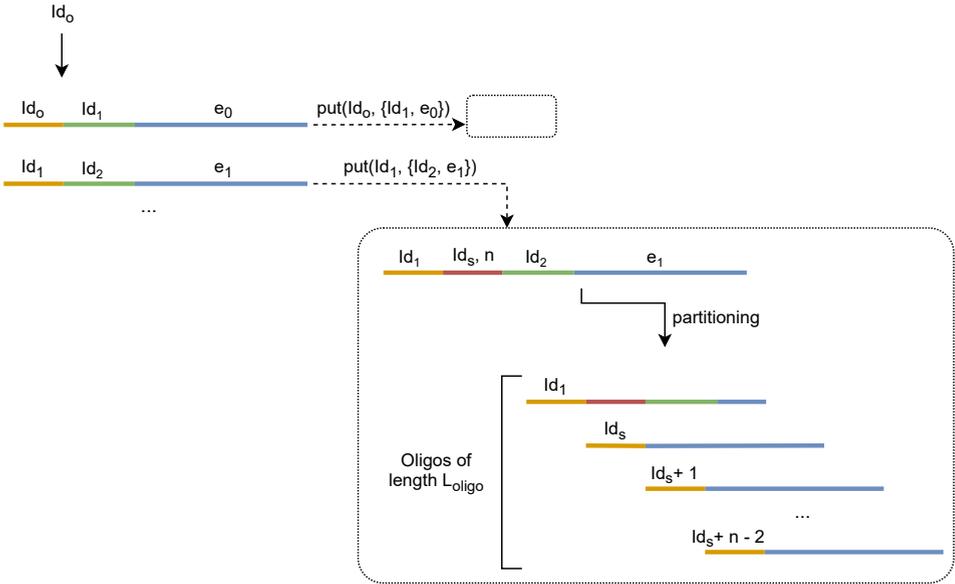


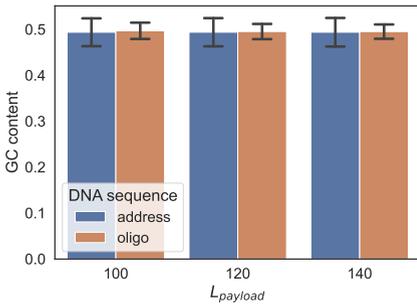
Fig. 6: Implementation of the list construct on DNAContainer.

To read an element from the list, we first call  $get(Id_o)$  to obtain the payload encoding  $Id_1$  and  $e_0$ . After that, we could return the first element  $e_0$ . Otherwise, we iterate through the list by sequentially calling  $get(Id_i)$ ,  $i = 1, \dots$  until we obtain the desired element.

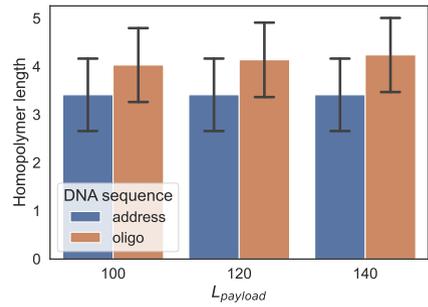
## 6 Experiments

We implemented DNAContainer in Java and tested it by simulating several million put and get operations. All the experiments were run on a computer with 256 logical cores (1.5 – 2.25 GHz each) and 1 TB of RAM. We used the data set from the OpenSky Network in ([https://opensky-network.org/datasets/publication-data/climbing-aircraft-dataset/trajs/A321\\_valid.csv.xz](https://opensky-network.org/datasets/publication-data/climbing-aircraft-dataset/trajs/A321_valid.csv.xz)), a relational table containing the tracking information of aircraft. We inserted the first 100,000 lines (records) into DNAContainer (without error correcting codes), varying the address and payload sizes, resulting in millions of oligos.

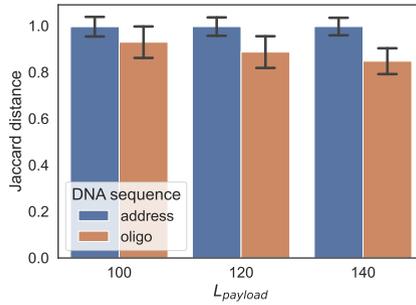
As shown in Fig. 7a, the obtained oligos' and DNA addresses' GC content is at  $\approx 50\%$ , adhering to the first constraint in Sect. 3.6. Furthermore, the average longest homopolymer's length is between 3 and 4, fulfilling the second required constraint as depicted in Fig. 7b. Moreover, to check the remaining constraints (3) and (4), we used LSH according to [El22] and set  $k = 5$  for the Jaccard similarity. Hence, our DNA addresses and oligos do not significantly overlap and fulfill the remaining constraints as shown in Fig. 7c.



(a) The GC content.



(b) The longest homopolymer length.



(c) The distance of every address and oligo to the other addresses and oligos calculated by LSH, respectively.

Fig. 7: GC content, the longest homopolymer length, and the mutual Jaccard distance of oligos and addresses.

The DNA optimization parameters are set as follows. For padding, we inserted  $\approx 16\%$  of the payload's size as padding information to each payload. Furthermore, we used  $m = 16$  permutations for DNA addresses and payloads. After optimization,  $L_{payload}$  ranged in 100, 120 and 140, and  $L_{address}$  in 60, and 80. Note that by increasing the number of permutations, we could, e.g., reduce the mutual overlaps of the oligos. However, setting the permutations count to  $m = 16$  was sufficient to obtain DNA without significant mutual overlaps. We repeated this experiment and turned off the permutations ( $m = 0$ ), resulting in longer homopolymers, and the largest difference was that the mutual overlaps increased significantly. In particular, the average mutual Jaccard distance of the DNA addresses dropped from  $\approx 1.0$  to  $\approx 0.5$ , and the average mutual Jaccard distance of the oligos dropped from  $\approx 1.0$  to  $\approx 0.75$ .

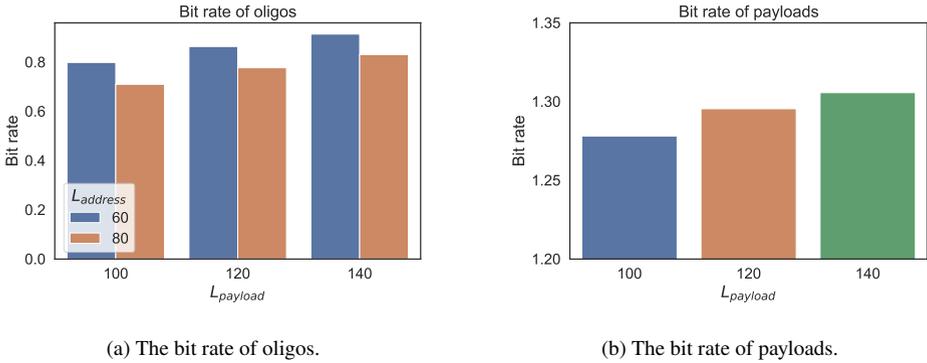
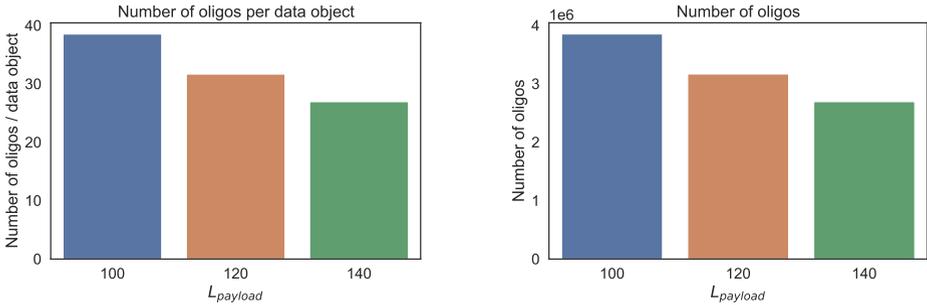


Fig. 8: The bit rate of oligos and payloads by varying the payload size.

Figure 8a depicts the *bit rate* of oligos, i.e., the information density for each oligo. The bit rate is calculated as the total number of digital bits divided by the total number of nucleotides. As expected, the bit rate for oligos with an address size of 80 is lower than that of 60 because the address does not encode any information. Fig. 8b presents the bit rate of payloads instead of oligos, i.e., ignoring the DNA address. The bit rate also increases by increasing the payload's size because more information can fit into the same number of oligos as depicted in Fig. 9a and Fig. 9b. Thus, by increasing the payload's size, we obtain fewer oligos encoding the same information. A bit rate of 2.0 bits/nucleotide being optimal, our system encodes the information to payloads at around 1.3 bits/nucleotide, which is a  $\approx 65\%$  capacity utilization of payloads. Since a payload of length  $L_{payload}$  optimally encodes two bits for every nucleotide and a single byte contains 8 bits, then the maximum capacity in bytes is calculated as:

$$\text{maximum capacity (bytes)} = 2 \cdot \frac{L_{payload}}{8} \quad (3)$$

For example, by setting  $L_{payload} = 140$ , the payload could encode up to 35 bytes. Plugging in our system's capacity utilization of 65% yields  $\approx 23$  bytes per payload. We repeated the same experiment above, stored the records in arrays and lists, and obtained similar results. We also tested turning on the Reed Solomon error correcting code, where the parameters are set such that up to 4 nucleotide erasures are corrected. The resulting capacity utilization was slightly lower at  $\approx 59\%$ . Despite that, DNAContainer manages to out-compete many recent DNA systems by supporting large-scale random access capabilities while maintaining a relatively high bit rate [Xu21, Do20, CNS19, Xu21].



(a) The number of oligos encoding a single data object.

(b) The number of oligos encoding all data objects.

Fig. 9: The number of oligos representing a single data object and the number of all oligos representing all the data objects.

Finally, to test the scalability of our approach, we translated 100 million addresses, i.e., we mapped 100 million Ids to corresponding DNA addresses (see Sect. 4.1) with  $L_{address} = 80$ , adhering to every constraint in Sect. 3.6. As shown above, our payloads with  $L_{payload} = 140$  reach a bit rate of 1.3, i.e., carry 23 bytes of information. Therefore, we could store up to  $23 \cdot 10^8$  bytes or 2.3 GB of information using these addresses, storing more information than the recently proposed DNA systems while providing sophisticated random access capabilities [Xu21]. Furthermore, only  $\approx 4000$  addresses were considered invalid of the generated 100 million addresses, and more addresses could be computed.

To store more information, e.g., in the terabyte range and beyond, in DNAContainer, more addresses must be generated, or a larger payload must be used. Current DNA synthesis technologies support synthesizing relatively short DNA sequences, whereas longer sequences are costly or not supported yet [HA19]. Additionally, we approximate the DNA overlaps using LSH, which requires extensive memory amounts. The computation time for generating the mentioned 100 million addresses took  $\approx 32.5$  hours with our computer, of which most of the time was spent on synchronizing checking constraint (3) for each generated permutation in parallel. Hence, the computation times could be significantly higher with a computer equipped with less memory or fewer processing cores.

Nevertheless, sequencing and especially synthesis technologies are constantly evolving. Certain synthesis technologies are developed, allowing the synthesis of several thousand nucleotides [Pi19]. For example, if we choose  $L_{payload} = 5,000$  and use  $10^{14}$  addresses, then the theoretical storage capacity of DNAContainer is  $\approx 11.5$  EB, which could be extended further by using more addresses or larger payloads.

## 7 Conclusion

This paper presents DNAContainer, an interface for DNA storage similar to a traditional storage device. DNAContainer offers an abstraction layer by providing simple put and get operations instead of synthesizing and sequencing DNA. Furthermore, we implement the common data structures array and list on DNAContainer, making data management more accessible. DNAContainer uses a virtual address space mapped to physical DNA addresses, facilitating random access to the data objects using traditional methods. Moreover, DNAContainer is aware of the required biochemical constraints. In particular, it encodes data objects as DNA oligonucleotides that are stable for long archival times and enables randomly accessing the data with the used virtual addresses. We tested our approach by simulating the insertion of several thousand data objects into DNAContainer, proving its scalability of managing up to millions of oligonucleotides addressed by millions of addresses.

In our future work, we will study multiple extensions of DNAContainer. In particular, we are interested in supporting more advanced index structures supporting expressive filter queries like range queries on DNA. Furthermore, we are designing a prototype of a physical DNA storage system where DNAContainer will be the primary interface.

## Acknowledgment

We thank the Hessian Ministry for Science and the Arts (LOEWE) for funding this work.

## Bibliography

- [AJ20] Appuswamy, Raja; Joguin, Vincent: Universal layout emulation for long-term database archival. arXiv preprint arXiv:2009.02678, 2020.
- [Al12] Allentoft, Morten E; Collins, Matthew; Harker, David; Haile, James; Oskam, Charlotte L; Hale, Marie L; Campos, Paula F; Samaniego, Jose A; Gilbert, M Thomas P; Willerslev, Eske et al.: The half-life of DNA in bone: measuring decay kinetics in 158 dated fossils. *Proceedings of the Royal Society B: Biological Sciences*, 279(1748):4724–4733, 2012.
- [Ap19] Appuswamy, Raja; Lebrigand, Kevin; Barbry, Pascal; Antonini, Marc; Madderson, Oliver; Freemont, Paul; MacDonald, James; Heinis, Thomas: OligoArchive: Using DNA in the DBMS storage hierarchy. In: *Biennial Conference on Innovative Data Systems Research (CIDR 2019)*. p. p98, 2019.
- [Ba20] Banal, James L; Shepherd, Tyson R; Berleant, Joseph D; Huang, Hellen; Reyes, Miguel; Ackerman, Cheri M; Blainey, Paul; Bathe, Mark: Random access DNA memory in a scalable, archival file storage system. bioRxiv, 2020.
- [Be15] Berlin, Konstantin; Koren, Sergey; Chin, Chen-Shan; Drake, James P; Landolin, Jane M; Phillippy, Adam M: Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623–630, 2015.

- [Bo16] Bornholt, James; Lopez, Randolph; Carmean, Douglas M; Ceze, Luis; Seelig, Georg; Strauss, Karin: A DNA-based archival storage system. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 637–649, 2016.
- [Br97] Broder, Andrei Z: On the resemblance and containment of documents. In: Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171). IEEE, pp. 21–29, 1997.
- [Bu01] Buhler, Jeremy: Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.
- [Bu13] Bumgarner, Roger: Overview of DNA microarrays: types, applications, and their future. *Current protocols in molecular biology*, 101(1):22–1, 2013.
- [Ch99] Chalikian, Tigran V; Völker, Jens; Plum, G Eric; Breslauer, Kenneth J: A more unified picture for the thermodynamics of nucleic acid duplex melting: a characterization by calorimetric and volumetric techniques. *Proceedings of the National Academy of Sciences*, 96(14):7853–7858, 1999.
- [CNS19] Ceze, Luis; Nivala, Jeff; Strauss, Karin: Molecular digital data storage using DNA. *Nature Reviews Genetics*, 20(8):456–466, 2019.
- [Co22] Cormen, Thomas H; Leiserson, Charles E; Rivest, Ronald L; Stein, Clifford: *Introduction to algorithms*. MIT press, 2022.
- [De90] Deux, O et al.: The story of O2. *IEEE Transactions on Knowledge & Data Engineering*, 2(01):91–108, 1990.
- [Do20] Dong, Yiming; Sun, Fajia; Ping, Zhi; Ouyang, Qi; Qian, Long: DNA storage: research landscape and future prospects. *National Science Review*, 7(6):1092–1107, 2020.
- [Du64] Durstenfeld, Richard: Algorithm 235: random permutation. *Communications of the ACM*, 7(7):420, 1964.
- [El22] El-Shaikh, Alex; Welzel, Marius; Heider, Dominik; Seeger, Bernhard: High-scale random access on DNA storage systems. *NAR genomics and bioinformatics*, 4(1):lqab126, 2022.
- [EZ17] Erlich, Yaniv; Zielinski, Dina: DNA Fountain enables a robust and efficient storage architecture. *science*, 355(6328):950–954, 2017.
- [GMM16] Goodwin, Sara; McPherson, John D; McCombie, W Richard: Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.
- [Go13] Goldman, Nick; Bertone, Paul; Chen, Siyuan; Dessimoz, Christophe; LeProust, Emily M; Sipos, Botond; Birney, Ewan: Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.
- [HA19] Heinis, Thomas; Alnasir, Jamie J: Survey of information encoding techniques for dna. *arXiv preprint arXiv:1906.11062*, 2019.
- [HMG19] Heckel, Reinhard; Mikutis, Gediminas; Grass, Robert N: A characterization of the DNA data storage channel. *Scientific reports*, 9(1):1–12, 2019.

- [IM98] Indyk, Piotr; Motwani, Rajeev: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613, 1998.
- [KC14] Kosuri, Sriram; Church, George M: Large-scale de novo DNA synthesis: technologies and applications. *Nature methods*, 11(5):499–507, 2014.
- [Ku01] Kurella, Manjula; Hsiao, Li-Li; Yoshida, Takumi; Randall, Jeffrey D; Chow, Gary; Sarang, Satinder S; Jensen, Roderick V; Gullans, Steven R: DNA microarray analysis of complex biologic processes. *Journal of the American Society of Nephrology*, 12(5):1072–1078, 2001.
- [Li20a] Li, Bingzhe; Song, Nae Young; Ou, Li; Du, David HC: Can We Store the Whole World’s Data in {DNA} Storage? In: 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20). 2020.
- [Li20b] Lin, Kevin N; Volkel, Kevin; Tuck, James M; Keung, Albert J: Dynamic and scalable DNA-based information storage. *Nature communications*, 11(1):1–12, 2020.
- [Li22] Lin, Yi-Syuan; Liang, Yu-Pei; Chen, Tseng-Yi; Chang, Yuan-Hao; Chen, Shuo-Han; Wei, Hsin-Wen; Shih, Wei-Kuan: How to Enable Index Scheme for Reducing the Writing Cost of DNA Storage on Insertion and Deletion. *ACM Transactions on Embedded Computing Systems (TECS)*, 21(3):1–25, 2022.
- [Ma20] Ma, Tian J; Garcia, Rudy J; Danford, Forest; Patrizi, Laura; Galasso, Jennifer; Loyd, Jason: Big data actionable intelligence architecture. *Journal of Big Data*, 7(1):1–19, 2020.
- [MFL14] Ma, Dongzhe; Feng, Jianhua; Li, Guoliang: A survey of address translation technologies for flash memories. *ACM Computing Surveys (CSUR)*, 46(3):1–39, 2014.
- [Or18] Organick, Lee; Ang, Siena Dumas; Chen, Yuan-Jyue; Lopez, Randolph; Yekhanin, Sergey; Makarychev, Konstantin; Racz, Miklos Z; Kamath, Govinda; Gopalan, Parikshit; Nguyen, Bichlien et al.: Random access in large-scale DNA data storage. *Nature biotechnology*, 36(3):242–248, 2018.
- [Pi19] Ping, Zhi; Ma, Dongzhao; Huang, Xiaoluo; Chen, Shihong; Liu, Longying; Guo, Fei; Zhu, Sha Joe; Shen, Yue: Carbon-based archiving: current progress and future prospects of DNA-based data storage. *GigaScience*, 8(6):giz075, 2019.
- [QSH22] Quah, Jasmine; Sella, Omer; Heinis, Thomas: DNA data storage, sequencing data-carrying DNA. *arXiv preprint arXiv:2205.05488*, 2022.
- [RS60] Reed, Irving S; Solomon, Gustave: Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.
- [Sc20] Schwarz, Michael; Welzel, Marius; Kabdullayeva, Tolganay; Becker, Anke; Freisleben, Bernd; Heider, Dominik: MESA: automated assessment of synthetic DNA fragments and simulation of DNA synthesis, storage, sequencing and PCR errors. *Bioinformatics*, 36(11):3322–3326, 2020.
- [Sp59] Spencer, M: The stereochemistry of deoxyribonucleic acid. I. Covalent bond lengths and angles. *Acta Crystallographica*, 12(1):59–65, 1959.

- [Wa19] Wang, Yixin; Zhang, Jingyun; Gunawan, Erry; Guan, Yong Liang; Poh, Chueh Loo et al.: High capacity DNA data storage with variable-length Oligonucleotides using repeat accumulate code and hybrid mapping. *Journal of biological engineering*, 13(1):1–11, 2019.
- [Xu21] Xu, Chengtao; Zhao, Chao; Ma, Biao; Liu, Hong: Uncertainties in synthetic DNA-based data storage. *Nucleic acids research*, 49(10):5451–5469, 2021.
- [YPFK06] Yakovchuk, Peter; Protozanova, Ekaterina; Frank-Kamenetskii, Maxim D: Base-stacking and base-pairing contributions into thermal stability of the DNA double helix. *Nucleic acids research*, 34(2):564–574, 2006.
- [Zh16] Zhirnov, Victor; Zadegan, Reza M; Sandhu, Gurtej S; Church, George M; Hughes, William L: Nucleic acid memory. *Nature materials*, 15(4):366–370, 2016.



# Accelerating Large Table Scan using Processing-In-Memory Technology

Alexander Baumstark<sup>1</sup>, Muhammad Attahir Jibril<sup>2</sup>, Kai-Uwe Sattler<sup>3</sup>

**Abstract:** Today's systems are capable of storing large amounts of data in main memory. Particularly, In-memory DBMSs benefit from this development. However, the processing of data from the main memory necessarily has to run via the CPU. This creates a bottleneck, which affects the possible performance of the DBMS. Processing-In-Memory (PIM) is a paradigm to overcome this problem, which was not available in commercial systems for a long time. However, with the availability of UPMEM, a commercial product is finally available that provides PIM technology in hardware. In this work, we focus on the acceleration of the table scan, a fundamental and memory-bound operation. We show and investigate an approach that can be used to optimize this operation by using PIM. We evaluate the PIM scan in terms of parallelism and execution time in benchmarks with different table sizes and compare it to a traditional CPU-based table scan. The result is a PIM table scan that outperforms the CPU-based scan significantly.

**Keywords:** UPMEM; Processing-In-Memory; In-Memory Database

## 1 Introduction

In-memory databases aim at low latency and high throughput for queries and updates in order to support real-time data processing. By keeping (most of) the data in main memory, workloads on such databases are typically memory-bound, and accessing main memory becomes more and more a bottleneck – a phenomenon that is known as memory wall [WM95]. However, novel and emerging memory technologies open up new opportunities such as offloading computation to memory. One example is Processing-in-Memory (PIM), a rather new concept where (simpler) operations can be executed directly in memory (on the same die) without moving the data from DRAM. The basic idea of this approach is to equip memory chips with additional processing units. Data can be processed directly on the memory chips without involving the system's CPU. PIM offers great potential: CPU load could be reduced, and memory bandwidth could be increased by reducing the amount of data to be transferred to the CPU.

Though, many PIM architectures have been proposed in the past (see [Ng20] for a classification), the only publicly available commercial product is offered by the UPMEM company. In addition, Samsung has also announced a product, but it is not available on the market yet. The

---

<sup>1</sup> TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, alexander.baumstark@tu-ilmenau.de

<sup>2</sup> TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, muhammad-attahir.jibril@tu-ilmenau.de

<sup>3</sup> TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, kus@tu-ilmenau.de

UPMEM technology has only recently become available (first presented at HotChips 2019), therefore, only a few experimental studies of UPMEM have been published. In [Ni21], the authors evaluate UPMEM PIM using a few use cases such as data compression, encryption, JSON processing, and text search. [G622] presents PrIM – the Processing-In-Memory benchmarks – a benchmark suite from different application domains as well as several key observations and programming recommendations. Though PrIM contains also a database selection operator, it is not integrated into a database engine. Both papers discuss also the technical details of UPMEM.

Based on these works, we investigate in this paper the potential of offloading data management processing to memory using PIM. Based on the observation that the performance of typical data management operations often depends on memory bandwidth or latency, we try to answer the question: *Can we accelerate in-memory operations in a database using PIM?* For this purpose, we use a graph database engine Poseidon<sup>4</sup>, which supports in addition to a persistent memory storage engine [Ji21] also an in-memory mode. However, because we focus in this paper on scan operations, the findings of our experiments are not limited to graph databases. Still, they can be generalized to scans on relational and other non-relational databases.

## 2 Related Work

PIM is a well-known technique to overcome the CPU-memory bottleneck for several decades. There have been a number of concepts and approaches to provide PIM on hardware since the 1990s [Pa97, Pa97, Dr02]. The high cost and lack of industrial support for this concept prevented the production and sale of real PIM hardware. Still, research was conducted based on prototypes. The PIM technology follows a similar approach to GPU processing. The design space of GPU-accelerated architectures transferred to PIM was investigated in the work of [Zh14]. Further, with LazyPIM, the authors of [Bo17] published a mechanism for reducing data exchange between CPU and PIM cores by means of caching. With the company around UPMEM, hardware providing real PIM-enabled DRAM DIMMs was published [UP22]. There are already a number of works concerning this architecture investigating its characteristics and applicability. Gomez-Luna et. al investigates the architecture for its limitations and performance as well as energy consumption [G622]. The result of the work shows that the UPMEM system achieves suitable performance as long as the individual components (DPUs) do not require communication (DPU-to-DPU). There is also available work concerning the applicability of PIM hardware on real use cases. [Gu] investigates the potential of PIM hardware for the acceleration of ML training. The results show that ML training using PIM hardware can improve the training process compared with GPU-based ML training. [Gi22] investigates the improvement of sparse matrix-vector multiplication using real PIM hardware. [Ka22] provided an efficient index data structure that leverages PIM.

---

<sup>4</sup> [https://dbgit.prakinf.tu-ilmenau.de/code/poseidon\\_core/-/tree/upmem](https://dbgit.prakinf.tu-ilmenau.de/code/poseidon_core/-/tree/upmem)

However, due to the short availability of real PIM hardware at the time of this work, there is, to our knowledge, no DBMS that directly integrates PIM.

### 3 PIM Technology

The first publicly available real-world PIM technology is provided by the UPMEM company [UP22]. Because our work is based on this technology, in the following we give a brief overview of this architecture and the programming model.

#### 3.1 UPMEM Architecture

The core of the UPMEM architecture is the UPMEM DIMMs, which are based on regular DDR4-2400 DIMM modules but equipped with additional PIM chips. A structural overview is given in Fig. 1. The UPMEM DIMMs are organized into ranks. A UPMEM DIMM consists of up to two ranks and each rank consists of up to 8 PIM-enabled chips. A PIM chip usually consists of 8 DRAM Processing Units (DPUs). Each DPU has exclusive access to 64 MB Main RAM (MRAM), 24 KB Instruction RAM (IRAM), and 64 KB Working RAM (WRAM) for processing. As DPUs have only access to their own MRAM there is no direct communication possible between different DPUs. Further, a DPU consists of a general-purpose 32-bit RISC core with a maximum achievable frequency of 400 MHz, which can execute a special instruction set in a multithreaded in-order pipeline. For multithreading, there are 24 hardware threads available. The context is switched on every cycle between the threads, which hides the memory latency [La16]. All threads share the same memory on the DPU which requires synchronization to guarantee consistency. This architecture allows the parallel execution of a program on different pieces of data directly on DRAM.

#### 3.2 Programming Model

For the utilization of the 24 hardware threads of a DPU, up to 24 tasklets can be used. This follows the Single Program Multiple Data programming model. All threads are executed with the same code but on different pieces of data. The number of used tasklets must be defined by the programmer at compile-time. As the MRAM and WRAM are shared among all tasklets on a DPU, the model provides synchronization primitives like mutexes, semaphores, barriers, and handshakes. Critical sections in the execution of a DPU program can be protected by mutexes with `mutex_lock` and `mutex_unlock` methods. Their effect is the same as the mutexes in usual systems. The critical section is then only accessible by one tasklet at a time. The purpose of the barriers is to control the execution flow of all tasklets. This can be done by using the `barrier_wait` method. The tasklets of the DPU wait at this

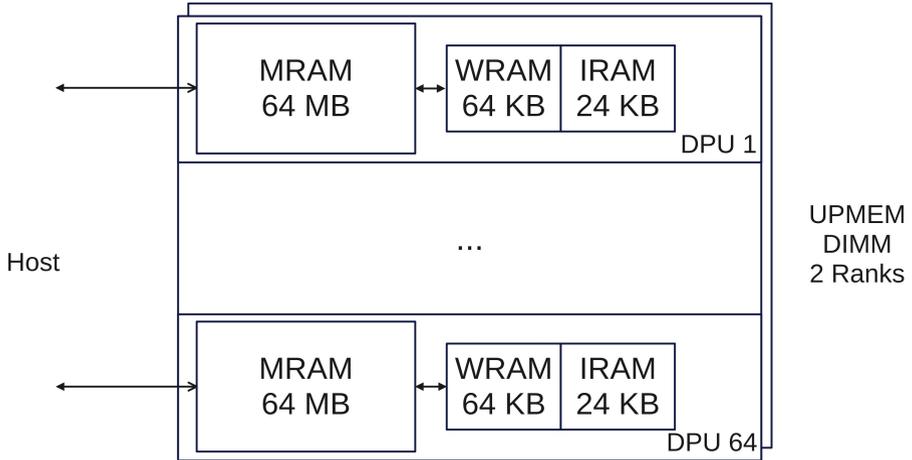


Fig. 1: Overview of a UPMEM DIMM with 2 Ranks of 64 DPUs.

point until all other tasklets have reached the barrier. After this, the execution continues. The handshake primitives are used for direct synchronization between the tasklets and the semaphores primitives for counters, similar to the counters known in operating systems. Using these primitives enables the effective utilization of multithreading provided by the DPUs.

The execution and control of the DPU program are handled by the host application. The host application allocates the set of desired DPUs and selects the appropriate DPU program. It is possible to allocate a specific rank or a specific number of DPUs. Further, the host application manages the execution of the DPU program and the data transfer to and from MRAM. The actual execution and data transfer can be handled synchronously and asynchronously by the host application. When executing the DPU program launch or the data transfer synchronously, the host application waits for the complete execution of the launch or data transfer. When transferring data, it is often desired to prepare the next batch for data transfer, while transferring the old batch. For this purpose, the UPMEM host library provides the possibility to execute the data transfer and DPU launch asynchronously. The asynchronous execution executes the instructions in the background using another thread and gives the control back to the host application. It allows the host application to proceed with the next batch for data transfer, or launch the same of another program on another set of DPUs.

The workflow of a host program running a DPU program with UPMEM technology can be summarized in the following steps:

1. Environment allocation (DPU, Ranks, DPU Program),
2. Buffer population from the host's main memory to MRAM of DPUs.

3. Execution of the DPU Program.
4. Retrieving of the processed results from the MRAM of the DPUs to the host's main memory.

The DPU program can be executed several times. The data is retained in the MRAM of the DPUs and does not have to be reinitialized. This is useful for tasks where a solution has to be calculated in several iterations. The DPU programs are written in the programming language C and compiled by a special compiler, which is based on LLVM and Clang.

### 3.3 Memory Management

As already mentioned, various memory types are available to a DPU. These differ in size and also in connection to the DPU. The largest memory available to a DPU is the MRAM. It has a capacity of 64 MB and has the purpose of exchanging data with the host. The host system can copy data from its main memory to the MRAM and also transfer data from the MRAM to the main memory of the host.

The WRAM of a DPU is a working memory in which a DPU stores the stack and global variables. Access to this memory is restricted to the DPU itself. Direct access from the host is not possible. Further, the DPU can access the WRAM only through 8-64 bit DMA instructions. The UPMEM runtime library provides for the transfer between MRAM and WRAM the methods `mram_read` for WRAM-MRAM and `mram_write` for MRAM-WRAM transfer. Each DMA instruction can copy up to 2 KB of data.

Communication with the host is done through data transfers between the main memory of the host and the MRAM of the DPU. The UPMEM runtime library provides different instructions for this purpose like `dpu_copy_to/dpu_copy_from` for copying a buffer from and to MRAM of specific DPUs. For parallel data transfer, the library provides the method `dpu_prepare_xfer` which assigns a buffer to a specific MRAM of a DPU. The actual data transfer is then performed in parallel using the `dpu_push_xfer` method but requires the same buffer size for all DPUs.

## 4 The Poseidon Graph Database

The present work is mainly developed for the graph database Poseidon. Although Poseidon was originally optimized for the characteristics of persistent memory, these characteristics can also be transferred to the exploitation of in-memory processing in DRAM. For this purpose, Poseidon already provides the necessary optimized data structures. In the following, the general architecture of Poseidon is described.

## 4.1 Data Model and Storage

The data layout of the Poseidon Graph Database is based on the labeled property graph model wherein labels and property values can be assigned to nodes and relationships. A complete graph in the Poseidon Graph Database consists of different tables in which the nodes, relationships, and respective property entries are stored. The tables are directly stored on DRAM. The Poseidon Graph Database provides also support for the storage of data directly on disk or Persistent Memory using similar data structures but optimized for utilization of the underlying storage. However, for the scope of this paper, we focus on the implementation of the storage on DRAM. For the underlying data structure, a linked list of fixed-size arrays (*chunks*) is used, which is referred to as a chunked vector. Furthermore, the nodes, relationships, and properties are stored in fixed-size records within the appropriate chunked vector. For entries with variable sizes, such as string values, an entry is created in a dictionary, and the corresponding dictionary code is stored in the respective record. To achieve the connection between nodes and relationships, the offsets of the respective entries are used. A node record stores the offsets of the first incoming and outgoing relationships. This offset points to an entry in the relationship table. A relationship record contains the offsets of the source and destination nodes. Moreover, the corresponding relationships are also linked to each other. A relationship record, therefore, contains the next offset of the relationship list of the source and destination node. The traversing through a graph can be achieved by alternately searching the node and relationship table for the offsets of the respective records.

## 4.2 Graph Queries

Processing graph database queries consists mainly of discovering a path between nodes in a graph. Besides the usual operators known from relational DBMSs like selections, projections, or joins, the Poseidon Graph Database provides an additional set of operators, especially for the processing of graph queries. These operators are based on graph algebra which is an extension of relational algebra [HG16]. For the data flow between operators, we implemented a push-based query processing approach. Here, the operators are organized into a pipeline and push their results toward the consuming operator until a pipeline breaker occurs [NL14]. For various reasons regarding the simplicity of a graph query language, we implemented an easy and manageable query language oriented to graph algebra.

```
Project({{0,"name"},{2,"name"}},  
Expand(OUT, "Person",  
ForeachRelationship(FROM, ":friendOf",  
NodeScan("Person"))))
```

Fig. 2: Example Graph Query in Poseidon

An example graph query in our language is given in 2. The aim of this query is to find all *Person* nodes in the graph, which are connected to another *Person* node with a *:friendOf* relationship, resulting in an overview of all friends in a graph.

The entry point of every query in Poseidon is the `NodeScan` operator. As the name suggests, it scans the underlying table for nodes, compares optionally each node with a given label, and pushes the appropriate nodes to the next operator. A scan as shown in the example is actually a scan combined with a filter for finding nodes with the given label. Because strings are dictionary encoded, this kind of filter is a simple integer comparison representing an appropriate candidate for offloading to UPMEM.

The nodes can be then processed with the `ForeachRelationship` operator, in order to find an ingoing or outgoing relationship of the previous node. Optionally, it compares the relationship label with a given label. The found relationship is then passed to the next operator. A relationship tuple can then be processed using the `Expand` operator. This operator extracts the source or destination node of the handed relationship. With these operators, it is possible to traverse a graph to find paths between two nodes.

### 4.3 Query Processing

For the processing of graph queries, Poseidon's query engine relies on push-based query processing and Morsel-driven parallelism [Le14]. The data flow at query processing is organized in a pipeline, and the resulting tuples are pushed from one operator toward their consuming operator. This flow continues until a pipeline breaker is reached. For parallelism, the engine exploits Morsel-driven parallelism using the underlying chunked vector data structure. Before the execution of the query, the query and each individual chunk of the chunked vector will be assigned to the task and pushed into a task pool. When executing the query, the engine spawns several threads which pull a task from the pool and executes the given query on this task until all tasks are processed.

The query engine provides three different execution modes for the actual processing: executing ahead-of-time (AOT) compiled code, just-in-time (JIT) compilation, and an adaptive approach. The AOT-compiled mode processes the given query using pre-compiled C++ methods, which execute the given operators. The JIT-compilation mode transforms the given graph query into highly optimized machine code and executes it directly. For this, we use the LLVM compilation framework. The graph query will be transformed into a single function in LLVM IR. Then, it will be optimized using several optimization passes like dead-code elimination or instruction combining. The resulting optimized LLVM IR code will then be transformed into machine code and executed by the engine. To hide compilation time, the engine can execute queries in the adaptive mode. Here, the engine starts the query processing using the AOT-compiled mode and compiles the query in the background. As soon as the compilation is complete, it switches to the new compiled code. Additionally, this

mode is useful to hide access latencies of the underlying storage type like disk or persistent memory [BJS21].

## 5 PIM-based Table Scans

The starting points of most queries are table scans. Often there is no other way than to traverse the entire table for tuples that match a given predicate. Especially in the case of predicates with particularly low selectivity, tuples that do not correspond to the predicate must be transferred unnecessarily via the CPU of the system. This procedure in today's usual systems leads to a bottleneck and reduces the possible performance. In this section, we will show the possibility of implementing a table scan operator by exploiting the PIM technology.

### 5.1 Memory layout

For the execution of table scans on the DPU, the memory of the DPUs must be taken into account according to their characteristics. The tables of nodes, relationships, and properties of the Poseidon Graph database are based on the chunked vector data structure. The chunking of the table can also be exploited for the design of the memory layout on the DPUs.

```
struct mram_node {
    uint8_t tx_pad[40];
    uint64_t id;
    uint64_t from_rship_list;
    uint64_t to_rship_list;
    uint64_t property_list;
    uint32_t node_label;
};
```

List. 1: Structure of node in MRAM

```
struct mram_chunk {
    struct mram_node data[C_ELEMENTS];
    struct mram_chunk* next;
    char bitset[BS_SIZE];
    uint32_t first;
    char padding[PAD_SIZE];
};
```

List. 2: Structure of chunk in MRAM

Listing 1 shows the structure of the nodes and Listing 2 is the structure of the chunks which are stored in MRAM. The required size of the nodes for storing the necessary data is 80 bytes. We leave the parts that are used for transactional processing out of the scope of this paper for the moment and label them as `tx_pad`. A similar structural layout is used to represent the relationships and properties in the MRAM of the DPUs. Basically, the representations are equivalent to those used for storing the data in the main memory of the host. In addition, care was taken that the size is a multiple of 8 bytes to allow transfer to MRAM and between MRAM-WRAM without additional transformation. The appropriate alignment of the data allows the direct transfer to the DPU but also buffering a part of the data in the DPU WRAM for faster access. With large tables, it may happen that more chunks exist than available DPUs. To use the memory of a DPU efficiently and to process as much

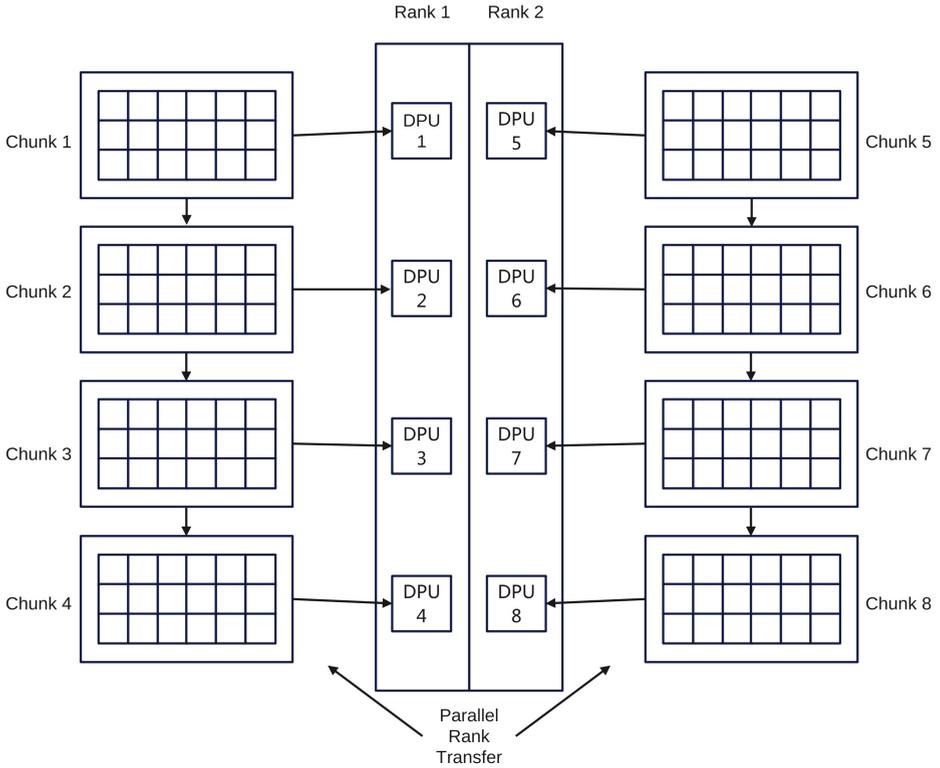


Fig. 3: Rank Parallel Chunk to DPU Assignment.

data as possible on it, it is more beneficial to transfer several chunks to a DPU. Considering the chunk size of 65536 bytes, we reserve in each DPU space that is able to hold up to 1000 chunks. The remaining MRAM space can be used for parameters such as the number of chunks passed, filter arguments, and storing the results.

## 5.2 Chunk-DPU Assignment

In order to transfer the data efficiently, we make use of asynchronous and parallel host-to-DPU data transfer. To implement the data transfer as efficiently as possible the data must be transferred as parallel as possible. This is achieved with DPU and Rank parallel data transfer. The underlying data structure in Poseidon, which is used for the storage of nodes and relationships, is perfectly suited to achieve this with the least possible implementation effort. Fig. 3 shows the rank parallel chunk-to-DPU assignment. Each chunk is assigned a DPU on a rank in a round-robin way. After the assignment is done, the data is transferred

in parallel per rank. This ensures that the workload on all DPUs is similar. Furthermore, multiple chunks can be assigned to a DPU to make efficient use of the available MRAM memory. If the table does not fit completely into the MRAM, the program must be executed with the already transferred part. Then the remaining part must be transferred back to the DPU. The following listing shows the algorithm for the chunk to DPU assignment.

```
foreach(chunk) {
  DPU_RANK_FOREACH(set, rank) {
    DPU_FOREACH(rank, dpu) {
      dpu_prepare_xfer(dpu, chunk);
      dpu_push_xfer(rank, DPU_XFER_DEFAULT, "mram_chunks",
        offset, CHUNKS_SIZE, DPU_XFER_ASYNC);
      calc_offset(); }}}}
```

List. 3: Host to DPU chunk transfer algorithm

The algorithm iterates over the available chunks of nodes, relationships, or properties. Then it iterates over the DPU of a rank. This is advantageous to allow efficient parallel data transfer, as the buffer for the data transfer must be the same size and write to the same offset address in the MRAM. Otherwise, the transfer would be serial.

### 5.3 DPU Scan

To enable an efficient multithreading scan of the chunks, we divide the workload among all available tasklets. For this, we distribute the elements to all available tasklets per chunk. Each tasklet thus works on an allocated area in each chunk assigned to the DPU. Then each tasklet iterates over the allocated area of the chunks. In each iteration, a record is checked for a given filter predicate. As soon as a record matches the predicate, the result is saved by setting the corresponding position of the record in the chunk to 1 in a bit vector. Per DPU there is a single bit-vector for each passed chunk. This tasklet design also has the advantage that no further synchronization mechanisms are necessary since each tasklet writes the result to its own memory area.

## 6 Evaluation

We use the Social Network Benchmark (SNB) dataset from the Linked Data Benchmark Council (LDBC) for the following benchmarks. This is an applicable benchmark to evaluate the performance of this approach in a graph DBMS. The used scale factor of the dataset is 1. We further restrict ourselves to the nodes of the dataset which we store in a nodes table in Poseidon. In total, the table contains 1.180.565 entries, which are stored in 1445 chunks in DRAM with 817 entries per chunk. For the scan query, we scan the node entries for nodes labeled as Post with a selectivity of 10%.

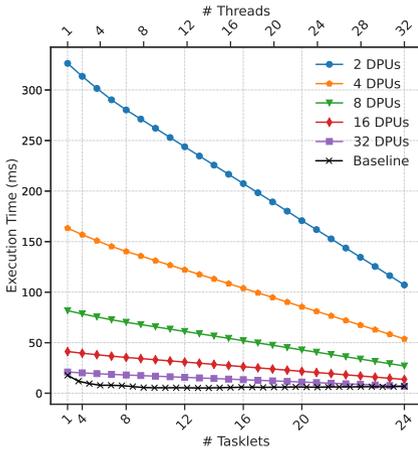


Fig. 4: Table Scan with 2 - 32 DPUs

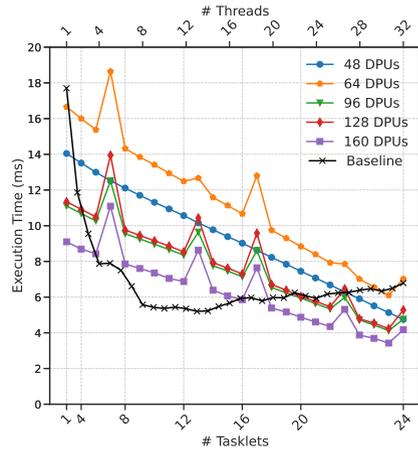


Fig. 5: Table Scan with 48 - 160 DPUs

## 6.1 System

The system used for the following benchmarks runs with two Intel Xeon Silver 4215R with a total of 16 cores with 2 threads each. A total of 32 threads can be executed on the system. Furthermore, the system has 512 GB of DRAM, which is made up of 8 x 64 GB DIMMs. In terms of PIM, the system has 4 UPMEM DIMMs with 16 GB each. Each UPMEM DIMM has 2 ranks with up to 64 DPUs each. The total number of DPUs is 510, divided into 8 ranks. The clock rates of the DPUs are between 200-400 MHz. The system runs under Ubuntu 20.04.1 with Linux kernel 5.4.0. The code of the host and DPU program was compiled with Clang at version 12 and full optimization at -O3.

The data layouts of the baseline and the DPU implementation are based on the same data structures and the same optimization to get a fair comparison.

## 6.2 DPU Parallelism

Fig. 4-Fig. 6 show the execution of table scans with different numbers of DPUs as well as with different numbers of tasklets. The baseline in these experiments is the usual CPU execution of the table scan with varying numbers of hardware threads (1-32). Furthermore, each thread performs the scan operation on the same number of chunks. The results in the baseline execution are saved in a result vector similar to the DPU program in order to obtain a workload as similar as possible. With a particularly large number of DPUs (164 or more), the execution runtimes change only slightly, since the size of the workload also changes only very slightly. The sweet spot for parallelism is around 12-24 task sets and a DPU count of around 160 for this table size.

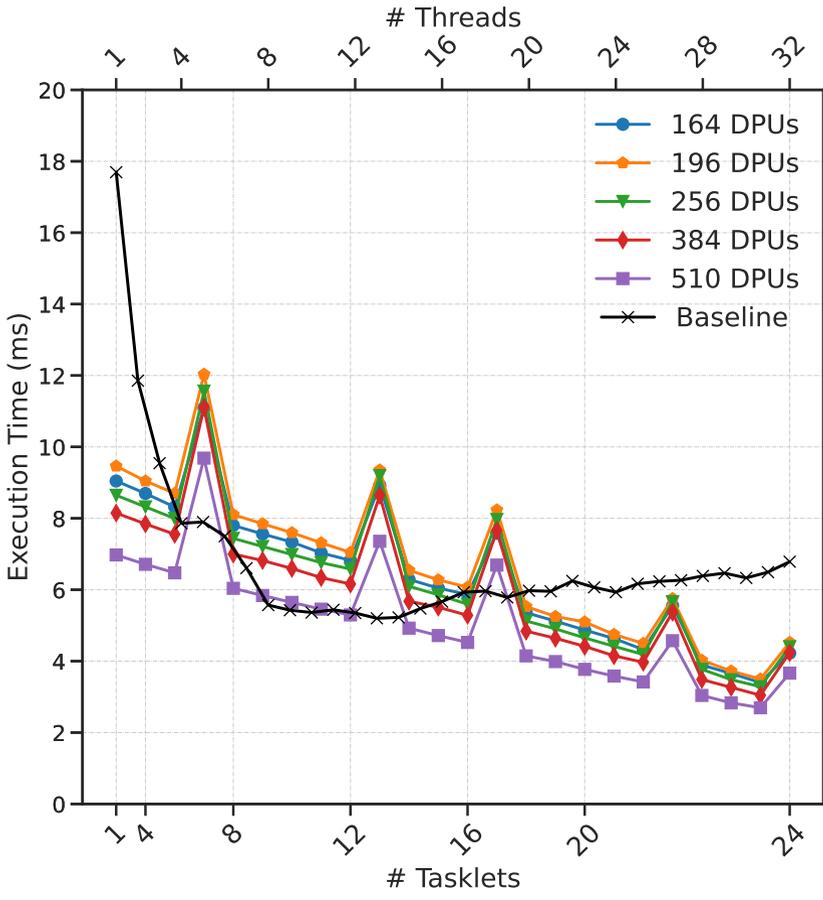


Fig. 6: Table Scan with 164 - 510 DPU

As the number of tasklets increases, the parallelism of the execution also increases. This can further improve the runtime. Furthermore, with an increasing number of DPUs the parallelism increases additionally. However, it can be seen that around 32 DPUs, which are used for the table scan, the runtime approaches the baseline of the CPU execution more and more. From 128 DPUs and the maximum number of 24 tasklets, the PIM execution is even faster than the CPU execution with all available hardware threads.

In summary, it can be concluded that the full utilization of the parallelism of the tasklets and a high number of DPUs can improve the runtimes of table scans by a considerable amount.

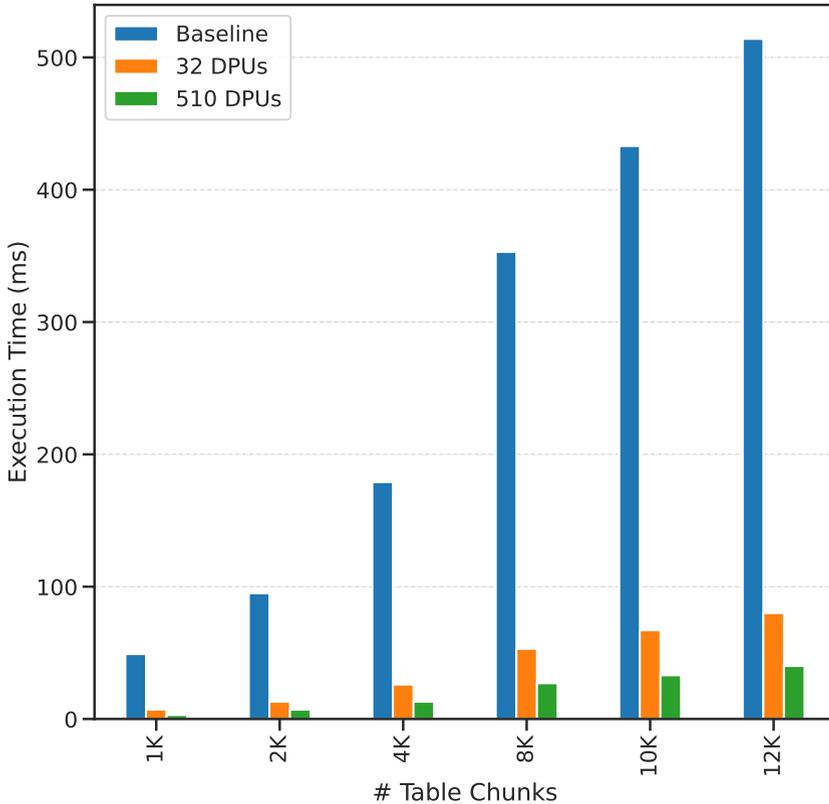


Fig. 7: Execution time of table scan on different table sizes.

### 6.3 Table Size

The execution times of the DPU scan with different table sizes are given in Fig. 7. The baseline of this experiment is again the execution of the table scan on the CPU with all available hardware threads. Each of these hardware threads executes the scan for several chunks. To achieve a similar workload, the baseline execution stores the result in a result vector, similar to the DPU program. The table size is represented by a different number of chunks. One chunk contains up to 817 records. A table that consists of 1000 chunks (1K) contains up to 817.000 entries. For this benchmark, we created an additional graph from using the LDBC SNB dataset containing 50% Post-nodes and 50% Person-nodes.

The linear increase in the execution time can be seen directly for all executions. Furthermore, the table scan on the DPU itself with a small number of 32 DPUs is much faster than the corresponding execution on the CPU with 32 threads. The high task parallelism can lead to very fast processing of the scan. Anyway, to enable the highest possible parallelism of the DPUs, it is necessary to have as little inter-DPU communication as possible and as little synchronization as possible at the tasklet level. In our approach, each tasklet worked on its own allocated memory space on the MRAM. Thus, no synchronization mechanisms were necessary. The result is a significant runtime improvement of the table scan.

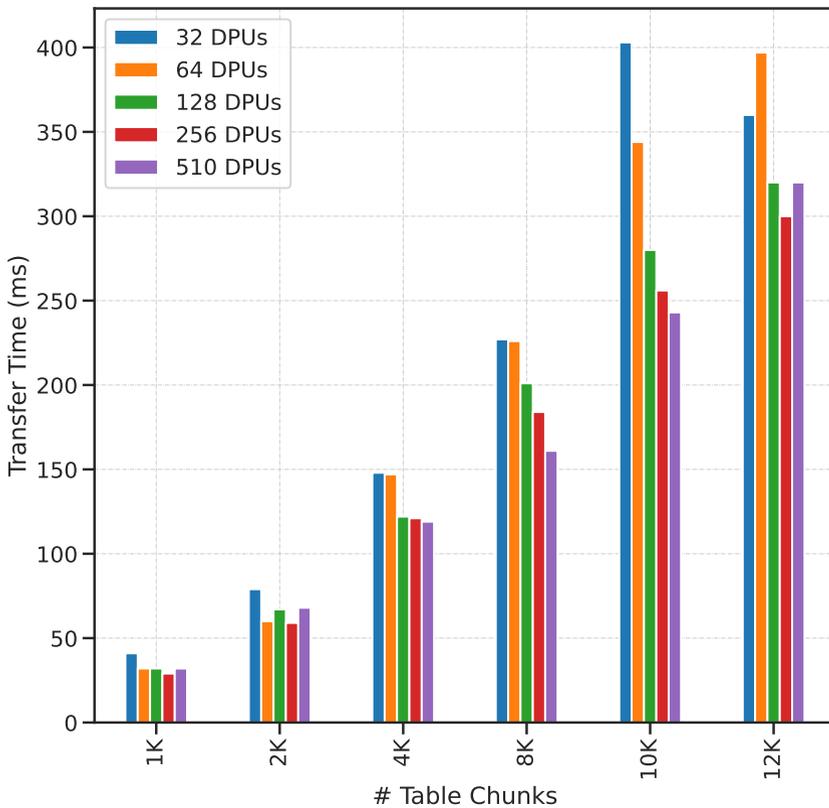


Fig. 8: Table transfer times from host to DPUs with different numbers of DPUs.

## 6.4 Data transfer

Fig. 8 shows the transfer times of large tables for different numbers of DPUs. For this experiment, we created several graphs with different numbers of chunks, ranging from 1000 (1K) to 12000 (12K). Each chunk contains up to 817 node records. We considered the transfer times on different numbers of DPUs to study the parallel data transfer. Furthermore, we made sure that the total number of chunks is distributed among the DPUs. If the amount of data does not fit into the memory of the DPUs, the data transfer would have to be executed multiple times. The results clearly show that the transfer times increase as the number of data increases. Parallel data transfer can reduce the transfer times by a few milliseconds. For example, the transfer time can be decreased by half if all 510 DPUs are used instead of 32.

Since the data only has to be loaded into the memory of the DPUs at the startup of the database, the result of the data transfer is acceptable. However, by using interleaved execution, the transfer times of data can be hidden. However, these possibilities are outside the scope of this paper.

## 7 Conclusion

The table scan is the most basic operator in the query processing of databases. In this work, we have investigated how we can accelerate table scans with filters using PIM technology. However, this approach requires an adapted design, since this new paradigm brings different characteristics with it, such as the transfer of data and the partitioning of the workloads in order to achieve the highest possible parallelism. As shown in the benchmarks presented here, PIM technology can outperform the runtime of a comparable CPU execution. To achieve this, however, high parallelism is needed. Furthermore, PIM technology can also be used to improve other operators. It is conceivable that especially the operators which are needed in graph databases for traversing can be further improved in their runtime.

Even when utilizing the CPU with its maximum possible parallelism, the results cannot come close to the runtimes of a table scan on multiple DPUs. With very large tables, this effect becomes even more pronounced. This has several implications for the execution of table scans. To save as much runtime as possible, as much data as possible, if not all of it, must be transferred to the DPUs' memory. A problem that comes along with this is data transfer. As shown in the evaluation, the transfer times increase with increasing table size. To take advantage of this as much as possible, the data must be transferred to the memory when the DBMS is started. The execution of updates and the maintenance of the consistency of this data is another problem, which is outside the scope of this paper. Furthermore, the data transfer can be optimized by using the possible bandwidth of the DPUs to transfer as much as possible in parallel.

The economic aspect of the currently available PIM hardware cannot be fairly measured and compared to the baseline hardware at the time of this work. Current hardware is currently only prototypically distributed by the UPMEM company.

## 8 Outlook

In future work, plan to integrate the compilation process of DPU programs directly into the query compiler. This would allow more flexible filter operations instead of only pre-coded filters. Furthermore, the design of an approach for asynchronous execution can be subject to investigation. The data transfer times are particularly high for very large tables. If this data were to be transferred to the DPUs before query execution, this would have a negative impact on the overall response time of the system. With an adaptive design, which transfers data asynchronously and executes it in parallel on different DPUs or on the rank level, this problem can be efficiently overcome. Finally, having multiple memory technologies including PIM available in a database server raises the question of data placement and/or efficient data transfer.

**Acknowledgements.** This work was partially funded by the German Research Foundation (DFG) in the context of the project “Hybrid Transactional/Analytical Graph Processing in Modern Memory Hierarchies (#TAG)” (SA 782/28-2) as part of the priority program “Scalable Data Management for Future Hardware” (SPP 2037), “Processing-In-Memory Primitives for Data Management (PIMPMe)” (SA 782/31) as part of the priority program “Disruptive Memory Technologies” (SPP 2377), and by the Carl-Zeiss-Stiftung under the project “Memristive Materials for Neuromorphic Electronics (MemWerk)”.

## Bibliography

- [BJS21] Baumstark, Alexander; Jibril, Muhammad Attahir; Sattler, Kai-Uwe: Adaptive Query Compilation in Graph Databases. In: 37th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2021, Chania, Greece, April 19-22, 2021. IEEE, pp. 112–119, 2021.
- [Bo17] Boroumand, Amirali; Ghose, Saugata; Patel, Minesh; Hassan, Hasan; Lucia, Brandon; Hsieh, Kevin; Malladi, Krishna T.; Zheng, Hongzhong; Mutlu, Onur: LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory. *IEEE Computer Architecture Letters*, 16(1):46–50, 2017.
- [Dr02] Draper, Jeff; Chame, Jacqueline; Hall, Mary; Steele, Craig; Barrett, Tim; LaCoss, Jeff; Granacki, John; Shin, Jaewook; Chen, Chun; Kang, Chang Woo; Kim, Ihn; Daglikoca, Gokhan: The Architecture of the DIVA Processing-in-Memory Chip. In: Proceedings of the 16th International Conference on Supercomputing. ICS '02, Association for Computing Machinery, New York, NY, USA, p. 14–25, 2002.

- [Gi22] Giannoula, Christina; Fernandez, Ivan; Gómez-Luna, Juan; Koziris, Nectarios; Goumas, Georgios; Mutlu, Onur: , Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems, 2022.
- [Gó22] Gómez-Luna, Juan; Hajj, Izzat El; Fernandez, Ivan; Giannoula, Christina; Oliveira, Geraldo F.; Mutlu, Onur: Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access*, 10:52565–52608, 2022.
- [Gu] Guo, Juan Gómez-Luna; Yuxin; Brocard, Sylvan; Legriel, Julien; Cimadomo, Remy; Oliveira, Geraldo F; Singh, Gagandeep; Mutlu, Onur: Machine Learning Training on a Memory-Centric Computing System.
- [HG16] Hölsch, Jürgen; Grossniklaus, Michael: An Algebra and Equivalences to Transform Graph Patterns in Neo4j. In (Palpanas, Themis; Stefanidis, Kostas, eds): *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016*. volume 1558 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
- [Ji21] Jibril, Muhammad Attahir; Baumstark, Alexander; Götze, Philipp; Sattler, Kai-Uwe: JIT happens: Transactional Graph Processing in Persistent Memory meets Just-In-Time Compilation. In (Velegrakis, Yannis; Zeinalipour-Yazti, Demetris; Chrysanthis, Panos K.; Guerra, Francesco, eds): *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. *OpenProceedings.org*, pp. 37–48, 2021.
- [Ka22] Kang, Hongbo; Zhao, Yiwei; Blleloch, Guy E; Dhulipala, Laxman; Gu, Yan; McGuffey, Charles; Gibbons, Phillip B: PIM-tree: A Skew-resistant Index for Processing-in-Memory. *arXiv preprint arXiv:2211.10516*, 2022.
- [La16] Lavenier, Dominique; Deltel, Charles; Furodet, David; Roy, Jean-François: *BLAST on UPMEM*. PhD thesis, INRIA Rennes-Bretagne Atlantique, 2016.
- [Le14] Leis, Viktor; Boncz, Peter A.; Kemper, Alfons; Neumann, Thomas: Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In (Dyreson, Curtis E.; Li, Feifei; Özsu, M. Tamer, eds): *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*. *ACM*, pp. 743–754, 2014.
- [Ng20] Nguyen, Hoang Anh Du; Yu, Jintao; Lebdeh, Muath Abu; Taouil, Mottaqiallah; Hamdioui, Said; Catthoor, Francky: A Classification of Memory-Centric Computing. *ACM J. Emerg. Technol. Comput. Syst.*, 16(2):13:1–13:26, 2020.
- [Ni21] Nider, Joel; Mustard, Craig; Zoltan, Andrada; Ramsden, John; Liu, Larry; Grossbard, Jacob; Dashti, Mohammad; Jodin, Romaric; Ghiti, Alexandre; Chauzi, Jordi; Fedorova, Alexandra: A Case Study of Processing-in-Memory in off-the-Shelf Systems. In (Calciu, Irina; Kuenning, Geoff, eds): *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. *USENIX Association*, pp. 117–130, 2021.
- [NL14] Neumann, Thomas; Leis, Viktor: Compiling Database Queries into Machine Code. *IEEE Data Eng. Bull.*, 37(1):3–11, 2014.
- [Pa97] Patterson, D.; Asanovic, K.; Brown, A.; Fromm, R.; Golbus, J.; Gribstad, B.; Keeton, K.; Kozyrakis, C.; Martin, D.; Perissakis, S.; Thomas, R.; Treuhaf, N.; Yelick, K.: *Intelligent RAM (IRAM): the industrial setting, applications, and architectures*. In: *Proceedings*

International Conference on Computer Design VLSI in Computers and Processors. pp. 2–7, 1997.

[UP22] UPMEM: , <https://www.upmem.com/>, 2022.

[WM95] Wulf, William A.; McKee, Sally A.: Hitting the memory wall: implications of the obvious. SIGARCH Comput. Archit. News, 23(1):20–24, 1995.

[Zh14] Zhang, Dongping; Jayasena, Nuwan; Lyashevsky, Alexander; Greathouse, Joseph L.; Xu, Lifan; Ignatowski, Michael: TOP-PIM: Throughput-Oriented Programmable Processing in Memory. In: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing. HPDC '14, Association for Computing Machinery, New York, NY, USA, p. 85–98, 2014.

# Enabling Integrated Data Analysis Pipelines on Heterogeneous Hardware through Holistic Extensibility

Extended Abstract (New Idea)

Patrick Damme<sup>1</sup> Matthias Boehm<sup>2</sup>

## 1 Introduction

Integrated data analysis (IDA) pipelines, that combine data management/query processing, high-performance computing, and machine learning training/scoring, become increasingly common in practice. Systems of these areas share many compilation and runtime techniques, and stress every hardware aspect of storage, computation, and networking. Accordingly, these systems are strongly impacted by hardware challenges such as the end of Dennard scaling and the end of Moore's law, which ultimately lead to dark silicon and increasing specialization at device level (CPUs, GPUs, FPGAs, ASICs), storage level (computational memory/storage, storage hierarchies), and workload level (data types and sparsity).

While this makes research on novel and heterogeneous hardware more exciting than ever, researchers are increasingly confronted with the question of how to integrate their prototypes to evaluate their impact on end-to-end IDA pipelines. Building yet another dedicated system offers a lot of flexibility, but requires substantial infrastructure efforts. However, enhancing an established system requires deep knowledge of the system internals and can be very hard. Thus, already in the 1980/90s, there was a wave of research on *extensible* DBMSs [CH90]. One of the most famous systems developed at that time is Postgres, which allows adding user-defined data types, functions, and access methods [SAH87]. Since then, concepts for *extensibility* and *variability* have been proposed for various system components, at different abstraction levels, and in different kinds of data systems. Recently, extensibility has also gained traction in the context of component-based systems [HD23]. However, to the best of our knowledge, there is no system infrastructure that *holistically* supports user extensions for all components relevant to the efficient execution of IDA pipelines on today's heterogeneous compute/storage hardware. To overcome this problem, we propose *holistic extensibility*.

In this talk, we present the concept of holistic extensibility for IDA pipelines, sketch how we approach this concept in DAPHNE, and provide an overview of our ongoing work.

---

<sup>1</sup> Technische Universität Berlin, Germany, patrick.damme@tu-berlin.de

<sup>2</sup> Technische Universität Berlin, Germany, matthias.boehm@tu-berlin.de

## 2 Holistic Extensibility for IDA Pipelines

*Holistic* extensibility means that every aspect of a data processing system for IDA pipelines should be easily extensible by users without a deep understanding of the system internals. This concept can be seen as an *ideal*, since it is hard to define a provably complete set of aspects requiring extensibility, and the need can evolve over time (e.g., integrating heterogeneous hardware was not a focus in the 1980s). However, we identify the following *extensibility aspects* relevant to the integration of novel computing or storage hardware:

**Operators.** Supporting different hardware accelerators typically requires dedicated operator code for each device. For instance, a CPU operator may be written in C++ employing SIMD intrinsics while a GPU operator may be written in CUDA. An extensible system should enable the integration of different physical operators, targeting different devices, for the same logical operator. Moreover, it should allow the definition of new (e.g., composite) operators in cases where this facilitates the execution on a particular device.

**Data Representation.** Operators targeted at specific hardware often *require* or *enable* specific data representations in terms of the overall storage layout (data types) and individual values (value types). E.g., in linear algebra for ML and simulations, different dense and sparse matrix data types were proposed. Furthermore, new value types for certain accelerators are emerging, e.g., `tf32` (GPUs) or `bf16` (TPUs). Moreover, specialized hardware often addresses specific applications, requiring the extension by domain-specific data representations.

**Optimization & Scheduling.** To make effective use of extensions, the system's optimizer must be able to reason about them, which requires an extensible internal representation (IR). The crucial decisions include when to use which custom operator and data representation, and how to place operators and data on the available (heterogeneous) computation and storage devices. For this purpose, it must be possible to add specific optimization passes to statically decide based on inferred data properties and system architecture, as well as to add specific runtime schedulers to make dynamic decisions which take the current execution behavior and system load into account. Both of these can benefit from custom cost models.

Typically, all of these extensibility aspects need to interact to fully integrate a novel hardware device. However, a *low barrier of entry* is crucial to achieve adoption and to facilitate *exploratory specialization*. For instance, it should be possible to add a physical operator for an accelerator without building an entire new processing engine. Moreover, existing operator implementations should be reusable for a custom data representation with acceptable out-of-the-box performance, to allow the user to focus on specializing and optimizing heavy-hitter operators for this representation. Finally, a deep integration into the optimizer should be *optional* by supporting *hints* on which physical variant and accelerator to use for an operator and which representation and storage device to use for an intermediate result.

The crucial aspects of holistic extensibility include: (1) how to balance expressiveness and additional complexity of the extensible system, and (2) how to achieve superb performance underneath the newly introduced abstractions.

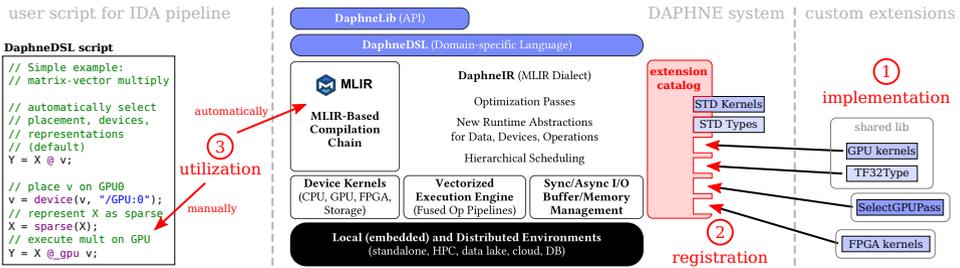


Fig. 1: DAPHNE System Architecture with Three-step Extension Approach.

### 3 Towards Holistic Extensibility in DAPHNE

DAPHNE<sup>3</sup> [Da22] is an open and extensible system infrastructure for IDA pipelines, including language abstractions, compilation and runtime techniques, multi-level scheduling, heterogeneous hardware accelerators, and computational storage for increasing productivity and eliminating unnecessary overheads. IDA pipelines are expressed in DaphneDSL, a domain-specific language for linear algebra and extended relational algebra over matrices and frames. DaphneDSL is parsed into DaphneIR. In an MLIR-based [La21] compilation chain, DaphneIR is optimized by domain-specific and traditional programming language optimizations, lowered to LLVM with calls to *pre-compiled* operator kernels, JIT compiled, and executed in a local or distributed runtime. DAPHNE’s *vectorized engine* fuses pipelines of operators, serves as the central means for parallelism, and is the central component for simultaneously utilizing heterogeneous hardware such as GPUs, FPGAs, and computational storage. Next, we give an overview of our extensibility design and mention some interesting research questions. To extend DAPHNE, users follow a three-step approach (Figure 1).

**1. Implementation.** The user implements the custom extensions for kernels, data/value types, optimizer passes, or scheduling techniques outside the DAPHNE code base in C++ adhering to well-defined *extension hooks*, and compiles them as a shared library. This does not require a deep understanding of the DAPHNE code base. Research questions include defining the right interfaces to balance expressiveness and complexity, achieving efficiency underneath these abstractions, and combining existing kernels and new data/value types.

**2. Registration.** The user registers the extension in DAPHNE’s *extension catalog* either through configuration files or from DaphneDSL. This requires providing the name and shared library as well as information specific to kernels (e.g., DaphneIR operation, expected input/output data/value types, required interesting data properties), data types (logical data type, preferred slicing axis for partitioning), and value types (bit width, semantics). More information can *optionally* be provided, e.g., traits and cost models to be used by the DAPHNE compiler. As the extension catalog can get large, its internal structure is decisive to efficiently serve relevant access patterns like look-up by operation and hardware device.

<sup>3</sup> <https://github.com/daphne-eu/daphne>

**3. Utilization.** By default, DAPHNE makes all decisions like the selection of kernels and physical data types as well as placement *automatically*, to increase users' productivity. To support this behavior for custom extensions, one option is to provide traits and cost models (e.g., for operator execution times or physical data size) in the extension catalog, which can be used by built-in optimization passes. Another option is to add a new optimization pass employing the extension where beneficial, which is simplified in DAPHNE due to the modular nature of the optimizer pipeline in MLIR. Even entire third-party MLIR dialects could be added, including operations, traits, and transforms. In fact, this is a promising option for generating code for hardware accelerators. This approach allows integrating new accelerators through dedicated dialects. Interesting questions include suitable abstractions for cost models and the integration of custom traits and interesting properties into the existing optimizer. To facilitate experimentation, DAPHNE also supports *manual* decisions. In DaphneDSL, users can provide *hints* on which device, kernel, or physical data representation to use for an operation or intermediate. These hints are treated as constraints by the optimizer. Interesting questions include the propagation of hints through the IR.

## 4 Conclusions and Outlook

We proposed *holistic extensibility* for IDA pipelines to handle increasing specialization from operators for heterogeneous hardware over the often co-designed data representations to the corresponding optimization and scheduling techniques. We sketched the extensibility design of DAPHNE, which offers users great benefits, while requiring low effort. We are currently implementing this design with a focus on kernels and data/value types, including some useful example extensions to showcase its simplicity. Our vision is to enable researchers to easily integrate their prototypes into a full-fledged system for IDA pipelines with minimal effort, thereby simplifying experimentation with and sharing of their work.

**Acknowledgments.**  The DAPHNE project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957407.

## Bibliography

- [CH90] Carey, Michael J.; Haas, Laura M.: Extensible Database Management Systems. SIGMOD Rec., 19(4):54–60, 1990.
- [Da22] Damme, Patrick et al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. In: CIDR. 2022.
- [HD23] Haffner, Immanuel; Dittrich, Jens: mutable: A Modern DBMS for Research and Fast Prototyping. In: CIDR. 2023.
- [La21] Lattner, Chris et al.: MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In: CGO. 2021.
- [SAH87] Stonebraker, Michael; Anton, Jeff; Hirohama, Michael: Extendability in POSTGRES. IEEE Data Eng. Bull., 10(2):16–23, 1987.

Workshop on Big (and Small) Data in Science and  
Humanities (BigDS)



# Using SQL/MED to Query Heterogeneous Data Sources with Alexa Voice Commands

Johannes Schildgen,<sup>1</sup> Florian Heinz,<sup>1</sup> Andreas Olijnyk,<sup>1</sup> Arvid Lindenau<sup>1</sup>

**Abstract:** Typical Alexa skills and other add-ons for voice assistants need to be custom developed for their one specific use case. This paper presents an approach to map arbitrary data sources (databases, APIs, services) to the relational model by using SQL/MED and to transform voice-based queries into SQL. The key challenges for such a universal skill are to correctly map the natural-language question into a SQL query on the correct source table in the federated database and to convert the result set back to a compact and well-understandable answer.

**Keywords:** Voice Assistants; SQL/MED; Natural-Language Processing; User interfaces for big data

## 1 Introduction and Motivation

Speech recognition and voice-based queries have been research topics for many years. While transforming speech into written text has basically reached a good-enough level [Hu17], the big challenge in building voice assistants is understanding what the user wants to know or to do. As there is no universal machine that understands all kinds of voice queries, companies develop individual voice applications for each specific scenario: finding train connections, news, stocks, weather, and so on. Alexa, Siri, and Google Assistant only support a limited set of built-in commands. Nevertheless, they often use a fallback method by doing a traditional web search with the full query and speaking out the text of the most relevant search result. As this approach is very error-prone and does not support customized queries, the typical approach is developing custom add-ons for voice assistants. For Alexa, these add-ons are called skills.

Each skill has to be individually developed by defining a list of example sentences, so-called intents. An intent contains zero or more slots. A slot is like a variable; it has a name and a data type. The skill developers must implement a when-this-then-that behavior of what to answer or do when which intent is called. For example, “Which are the meals in the canteen on <date>”. Voice frameworks work in a flexible way so that this intent also matches the following input sentence: “Tell me today’s meals in the canteen.” In this case, the slot <date> will be set to the current date.

Besides these end-user-focused use cases, voice interfaces are increasingly adopted in professional contexts. A recent trend is the use of natural-language interfaces to work with

---

<sup>1</sup> OTH Regensburg, Postfach 120327, 93025 Regensburg, Germany  
{johannes.schildgen,florian.heinz}@oth-regensburg.de; {andreas.olijnyk,arvid.lindenau}@st.oth-regensburg.de

data-analytics applications [Go20]. These so-called conversational analytics allow users without data-science skills to explore and analyze data. Big business-intelligence platforms like Tableau Ask Data [Ma18] or SAP Conversational AI have been launched in recent years to enable companies with business data to use natural-language queries.

This paper presents an approach that uses the database language SQL as an intermediate layer between Alexa skills and arbitrary data sources. Of course, this approach is not limited to Amazon Alexa; it also works for arbitrary other voice assistants. Our command from above would be translated into the following query: `SELECT name FROM meals WHERE meal_date = current_date;` The table names and column names (`meals`, `name`, `meal_date`) are determined by exploring the database metadata. If the desired data is stored in the tables of a relational database, then this query can simply be sent to that database. If not, our approach uses external tables as proposed by the SQL/MED standard (Management of External Data) [Me01]. Depending on the federated database management system, this concept is also called foreign-table wrappers (PostgreSQL), datalinks (DB2), external tables (Oracle), connect tables (MariaDB) or virtual schemas (Exasol). The idea of SQL/MED is to virtually integrate external data sources, like APIs or NoSQL databases, and let them appear in the database catalog like native tables. Each query on these tables is forwarded to the external system on demand (see Figure 1). In the example above, `meals` could be an external table to the API of OpenMensa.org or a view that joins physical and external tables.

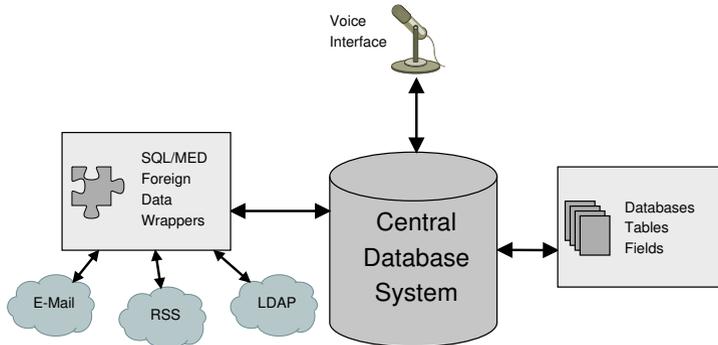


Fig. 1: Database System as the central data source for natural language queries

In this paper, we make the following contributions:

1. We present an SQL/MED-based approach for voice queries on arbitrary data sources.
2. We identify how to transform natural-language queries into valid SQL commands.
3. We transform query results back into a compact and well-understandable voice answer.

The following section shows a brief history of natural-language queries and other related work. In Section 3, we explain our approach for a universal Alexa skill. In Section 4, we present exemplary use cases and a short evaluation. We conclude in Section 5.

## 2 Related Work

Translating natural language into SQL queries (NL2SQL) has been a research topic for several decades [ART95, LJ14a, LJ14b]. Kim et al. [Ki20] provide an overview of those NL2SQL techniques. They use different benchmarks and automatic and manual matching approaches to evaluate whether natural-language commands are translated into a semantically correct SQL query or not. Some of the systems with the highest accuracy are NSP [Iy17], TypeSQL [Yu18], GNN [BGB19], and IRNet [Gu19]. They all use deep-learning methods to learn cross-domain as well as domain-specific terminologies and relationships. The approach that we used for our prototype implementation uses simple natural-language processing techniques. But it is easy to replace these techniques with others to increase the accuracy.

EchoQuery [Ly16] is a system that focuses on voice inputs and voice outputs to query relational databases. It can be used in an interactive way. The user can refine previous queries with follow-up questions, and the system can ask clarifying questions back to the user. The findings from this approach can easily be applied to our approach as well to make it more interactive. Cyrus [GJ19] is an iPhone app that converts voice commands into SQL queries for teaching purposes. The app can be used to learn SQL. Cyrus is very similar to EchoQuery and also to our approach. But our approach is voice-only; Cyrus displays the query and the result table on the smartphone display. They do not convert the result set into a well-understandable spoken answer.

There is only a little research on one-size-fits-all conversational systems. Haase et al. [Ha17] convert voice commands into SPARQL queries and send them to the Wikidata knowledge graph. However, most skills for Alexa and others are custom-developed for just one single application scenario. Atefi et al. [At20] studied the user reviews of more than 2800 of those custom Alexa skills. They found out that the most frequent complaint is that the content provided by the skill is undesired or uninteresting. The main reason for this is that in custom skills, users often need to ask questions in a prespecified and fixed form. With our approach, we try to solve this problem by building a universal Alexa skill. There, fuzzy matching methods on database metadata allow for a broad diversity of supported queries.

Given that databases are often accessed via an API, some work has been done on querying APIs with natural language (NLI2API). This is often done by training machine-learning models on examples of natural-language commands and their mapping to API calls. In [Su17], the authors proposed a framework to collect high-quality training data and evaluated it using real-world APIs, yielding good results. In follow-up work, an NLI2API system with fine-grained control was tested in a user study [Su18]. The participants had to complete several tasks using text queries, e.g., retrieving emails from their inboxes. The novel approach in this work was to split the queries into so-called modules that correspond to different API parameters. After performing a query, users could remove or add modules to correct their initial query. Eventually, the approach reduced the number of required interactions and task completion time and thus improved the overall user experience.

### 3 Implementation based on Amazon Alexa

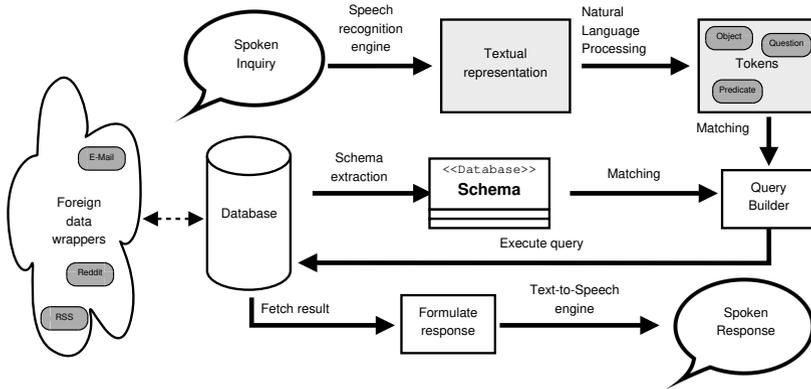


Fig. 2: Process Overview

Figure 2 sketches the coarse overview of the process. The human interface is a speech-recognition service as is provided by, for example, Alexa, Siri, Google Assistant, Cortana, Bixbi, or similar engines. Our reference implementation is basically independent of the speech-recognition service, but we chose Amazon Alexa for our prototype. The Alexa skill transcribes the spoken query into a text string, and this string is fully sent to our Python-based server via HTTPS. The only task of the voice service is the speech-to-text and text-to-speech part.

The main part of the process happens within a server application. When this server starts up, it first connects to the database to discover the database schema. In this step, not only the table and column names are fetched, but also a set of aliases for each of these identifiers is generated. They contain different word variations such as singular/plural forms or synonyms. For this task, string-normalization techniques, thesauruses, dictionaries, and the Python libraries “nltk”, “wordnet” and “inflect\_engine” are used.

After receiving the inquiry string, also this query is cleaned and normalized. For example, Alexa passes numbers up to ten as words, not as a sequence of digits, which is fixed up in that step, together with several other minor corrections like resolving abbreviations (“who’s” → “who is”) or question words (“how much” → “price”). These words are then used to try to find the correct parts of the database schema in the following order: (1) direct matches (original identifiers), (2) fuzzy search (using doublemetaphone), (3) matching of aliases, (4) association matching. After that, the matching concepts of the found tokens in the database query have to be determined: table name, column name (projection or selection), column value, and aggregation function. This is performed by heuristics that take into account the position inside the sentence, preceding or following words, and several more criteria. Finally, the query is executed and a natural sounding answer is generated from the result set. So, for example, the following conversations could be happening:

- > What is the firstname of the employee with employeeid five?
- The firstname of the employees with the employeeid 5 is 'Steve'.

It is quite useful that the answer always contains parts of the question. If the answer would just be “Steve”, the user cannot be that sure that the question was understood correctly by the system.

**Limitations** Our prototype supports single-table queries with multiple projection columns in the SELECT clause, simple aggregation functions (COUNT(\*), SUM|AVG|MIN|MAX(col)) without GROUP BY, and a WHERE clause with one or many (AND|OR) predicates. Each predicate consists of a column identifier, a comparison operator (<, <=, =, !=, >, >=, LIKE) and a numeric or string literal. These are the same limitations as in the WikiSQL [ZXS17] benchmark. For more complex queries with joins or groupings, the workaround is to create a view and query that view with a voice command.

## 4 Use Cases

In the previous section, we presented a simple implementation of how a natural-language query can be translated into a SQL query. This allows for voice queries on arbitrary tables in a relational database. This section describes some interesting use cases and evaluates the overall usability of this system. We will not only use native SQL tables for this but we will also use external tables. The idea is to virtually integrate external data so that it can be accessed with SQL. For our prototype, we use PostgreSQL’s foreign data wrappers. One can develop their own wrapper or use one of the available foreign data wrappers for JDBC, NoSQL (MongoDB, Neo4J, Redis, ...), files (CSV, JSON, XML, ...), services (Google Spreadsheet, Open Weather Map, Open Street Map, Twitter, ...) and many more<sup>4</sup>.

One useful foreign data wrapper is the *FileGW* module, which spans a bridge between the database and a local file system. After creating the foreign table, a typical command could be: “What is the content of the file with the name memo?”.

Another interesting foreign-data wrapper is the RSS gateway<sup>5</sup>:

```
CREATE FOREIGN TABLE newsfeed (title varchar(800), description varchar(800),
pubDate timestamp, link varchar(800)) server rss_srv options (url '...xml');
```

A command like “What are the titles of the newsfeed of the last two hours?” or “What is the description of the newsfeed with the title ‘clean energy?’” can now be used to extract information from arbitrary RSS feed sources.

Many types of information can be represented by a relational schema and the foreign-data-wrapper method is an easy and practical way to not only retrieve that information from a database but also link it with other relations in the database. It is for example possible to create a view that joins several (native or foreign) tables and retrieve information from this view with a spoken inquiry.

<sup>4</sup> [https://wiki.postgresql.org/wiki/Foreign\\_data\\_wrappers](https://wiki.postgresql.org/wiki/Foreign_data_wrappers)

<sup>5</sup> <https://multicorn.org/foreign-data-wrappers/>

External tables in PostgreSQL are not read-only but also writable. This makes it possible to extend our voice assistant by supporting inserts, updates, and deletes. For example, the foreign data wrapper for Philips Hue<sup>6</sup> shows a table with all lightbulbs and their states. A voice command can create an UPDATE query to turn on or off one or multiple lights.

Companies are increasingly adopting NLP technologies to power data-driven use cases. Decision makers use KPIs to monitor the current status of manufacturing processes, make analyses like trend predictions or get notified in case of critical deviations. We used our voice-based approach in a proof-of-concept implementation at an automotive company and it showed that it can improve the usability of a data-analytics application in a manufacturing domain. As an alternative to using the web application, users can use voice commands to query an underlying API and retrieve the desired information like “What is the current production status in plant Berlin?”. There has been a major challenge in implementing these use cases. Professional environments often have a domain-specific vocabulary that is difficult to integrate into NLP systems. Thus, it is important that the relevant terms are present in the database metadata. Database views can be used to structure the data in a useful way and to name the attributes accordingly, e.g., `production_status`. Once the data is structured in a useful way, the system can be used immediately.

Tests with users showed that the system improves the usability of the underlying KPI system because the information can be easily retrieved in any scenario directly on their phone without the need to have access to a PC.

## 5 Conclusion

This work shows a general-purpose method to retrieve specific information from an SQL database system. The focus where this work excels is to cope with database systems by only retrieving the database schema; other information is not needed. Nevertheless, the user can intuitively formulate spoken inquiries and the system often does a good job of translating the sentence into an SQL query. For this, fuzzy matching with Doublemetaphone and thesaurus lists is used to find out where the sought information might be stored. External information can be provided to the database by foreign-data-wrapper modules, where a rich amount of existing modules is available, but also developing customized wrappers is not a big task.

Further development should include the automatic joining of tables to retrieve information that cannot be derived from a single table. In the current state of the implementation, one or more views would have to be prepared in advance to achieve such a task. Beyond that, it would be desirable to support some more typical operations and aggregations on both sides - projections and predicates - of a query. As modern AI language models are well-suited for generating SQL queries, one approach could be integrating systems like OpenAI GPT to support more complex queries.

Our prototype’s source code is open-source, available at [github.com/fwheinz/datalexa](https://github.com/fwheinz/datalexa)

---

<sup>6</sup> <https://github.com/rotten/hue-multicorn-postgresql-fdw>

## Bibliography

- [ART95] Androutsopoulos, Ion; Ritchie, Graeme D; Thanisch, Peter: Natural language interfaces to databases—an introduction. *Natural language engineering*, 1(1):29–81, 1995.
- [At20] Atefi, Soodeh; Truelove, Andrew; Rheinschmitt, Matheus; Almeida, Eduardo; Ahmed, Iftekhar; Alipour, Amin: Examining user reviews of conversational systems: a case study of Alexa skills. *arXiv preprint arXiv:2003.00919*, 2020.
- [BGB19] Bogin, Ben; Gardner, Matt; Berant, Jonathan: Representing schema structure with graph neural networks for text-to-SQL parsing. *arXiv preprint arXiv:1905.06241*, 2019.
- [GJ19] Godinez, Josue Espinosa; Jamil, Hasan M: Meet Cyrus: the query by voice mobile assistant for the tutoring and formative assessment of SQL learners. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. pp. 2461–2468, 2019.
- [Go20] Goasduff, L. , Megatrends Dominate the Gartner Hype Cycle for Artificial Intelligence, 2020.
- [Gu19] Guo, Jiaqi; Zhan, Zecheng; Gao, Yan; Xiao, Yan; Lou, Jian-Guang; Liu, Ting; Zhang, Dongmei: Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.
- [Ha17] Haase, Peter; Nikolov, Andriy; Trame, Johannes; Kozlov, Artem; Herzig, Daniel M: Alexa, Ask Wikidata! Voice Interaction with Knowledge Graphs using Amazon Alexa. In: *ISWC (Posters, Demos & Industry Tracks)*. 2017.
- [Hu17] Huang, Xuedong: Microsoft researchers achieve new conversational speech recognition milestone. Microsoft, August, 2017.
- [Iy17] Iyer, Srinivasan; Konstas, Ioannis; Cheung, Alvin; Krishnamurthy, Jayant; Zettlemoyer, Luke: Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*, 2017.
- [Ki20] Kim, Hyeonji; So, Byeong-Hoon; Han, Wook-Shin; Lee, Hongrae: Natural language to SQL: where are we today? *Proceedings of the VLDB Endowment*, 13(10):1737–1750, 2020.
- [LJ14a] Li, Fei; Jagadish, Hosagrahar V: Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.
- [LJ14b] Li, Fei; Jagadish, Hosagrahar V: NaLIR: an interactive natural language interface for querying relational databases. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. pp. 709–712, 2014.
- [Ly16] Lyons, Gabriel; Tran, Vinh; Binnig, Carsten; Cetintemel, Ugur; Kraska, Tim: Making the case for Query-by-Voice with EchoQuery. In: *Proceedings of the 2016 International Conference on Management of Data*. pp. 2129–2132, 2016.
- [Ma18] Markas, Ruhaab. , *Ask Data: Simplifying analytics with natural language*, 2018.
- [Me01] Melton, Jim; Michels, Jan-Eike; Josifovski, Vanja; Kulkarni, Krishna; Schwarz, Peter; Zeidenstein, Kathy: SQL and management of external data. *ACM SIGMOD Record*, 30(1):70–77, 2001.

- [Su17] Su, Yu; Awadallah, Ahmed Hassan; Khabsa, Madian; Pantel, Patrick; Gamon, Michael; Encarnacion, Mark: Building natural language interfaces to web apis. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. pp. 177–186, 2017.
- [Su18] Su, Yu; Hassan Awadallah, Ahmed; Wang, Miaosen; White, Ryen W: Natural language interfaces with fine-grained user interaction: A case study on web apis. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. pp. 855–864, 2018.
- [Yu18] Yu, Tao; Li, Zifan; Zhang, Zilin; Zhang, Rui; Radev, Dragomir: Typesql: Knowledge-based type-aware neural text-to-sql generation. arXiv preprint arXiv:1804.09769, 2018.
- [ZXS17] Zhong, Victor; Xiong, Caiming; Socher, Richard: Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103, 2017.

# Integrating Access to Authority Data for Improved Interoperability of Research Data in the Digital Humanities

Robin Jegan<sup>1</sup>, Leon Fruth<sup>1</sup>, Tobias Gradl<sup>1</sup>, Andreas Henrich<sup>1</sup>

**Abstract:** Authority data is used to unambiguously identify persons, organizations and places. In this paper, a means to integrate access to several providers of authority data into data curation processes is described, which facilitates disambiguation of geographic data. Combined access to general datasets, in our case the Integrated Authority File (GND), as well as highly specialized datasets, here the Memorial Archives, improves the resolution of ambiguities and particularly benefits use cases of the Digital Humanities. The integration is necessary in order to abstract from technical, syntactical and semantic heterogeneity of the providers. Operations such as querying geographic information and receiving enriched data from different data sources are facilitated. An overview of the goals of the system, related projects and authority data providers are presented, as well as details on the implementation and further steps.

**Keywords:** authority file; geographic databases; data integration

## 1 Introduction

FAIR<sup>2</sup> has become a prominent keyword in academia that summarizes fundamental requirements of research data management. Authority files play a key role with respect to multiple FAIR principles in that entities such as persons, organizations and places can be unambiguously identified. Potentially ambiguous textual descriptions of entities (i.e. name attributions) can thus be replaced with references to their representation in authority files – improving the interoperability and in consequence the findability and reusability of data.

Numerous sources of authority data exist and provide access in terms of distinct data structures and formats – either by means of downloadable archives or in the form of accessible APIs. Data selection and their semantic and structural representation are influenced by requirements and contexts of respective providers. Authority files of national libraries, such as the Integrated Authority File (Gemeinsame Normdatei, (GND)<sup>3</sup> of the German National Library (DNB) aggregate authority data within their legal setting<sup>4</sup> and thus with a national bias. In contrast, universal data sources such as GeoNames<sup>5</sup> might expose a

---

<sup>1</sup> Otto-Friedrich-Universität Bamberg, Lehrstuhl für Medieninformatik, An der Weberei 5, 96047 Bamberg, Germany, [robin.jegan,leon.fruth,tobias.gradl,andreas.henrich]@uni-bamberg.de

<sup>2</sup> Findability, Accessibility, Interoperability, and Reuse of digital assets <https://www.go-fair.org/fair-principles/> All links accessed on 18-01-2023.

<sup>3</sup> [https://www.dnb.de/EN/Professionell/Standardisierung/GND/gnd\\_node.html](https://www.dnb.de/EN/Professionell/Standardisierung/GND/gnd_node.html)

<sup>4</sup> [https://www.gesetze-im-internet.de/dnbg/\\_2.html](https://www.gesetze-im-internet.de/dnbg/_2.html)

<sup>5</sup> <https://www.geonames.org/>

reduced descriptive depth. As an example of a highly specific normative data source, the Memorial Archives<sup>6</sup> contain entity descriptions in the specific context of the Flossenbürg Concentration Camp and as such, entities that are often unavailable in more generic authority files.

Concurrent access to multiple authority files is required if information of universal and specific files need to be referenced and combined. Use cases with such integrative requirements are often found in the Digital Humanities (DH), where historic entities and attributes such as names and chronologies are of particular relevance. As an example, the project Oral-History.Digital (OH.D)<sup>7</sup> develops a curation and research platform for collections of audiovisually recorded narrative interviews. Metadata and transcripts of interviews usually include geographic data such as historic places connected to the interviewed person or the location the interview was recorded. In order to clearly label and identify the spatial information, authority file providers are used to differentiate between ambiguous places. As such, interviews with survivors of the Flossenbürg concentration camp might contain references to specific places such as satellite camps, which can be found in the Memorial Archives. On the other hand, that same interview might reference commonly known places such as cities and buildings, that can be referenced by using the GND.

In this paper, we present an idea and software component that mediates between the required contextualization of research data (such as interviews) and the wide-spread authority data that is available. After presenting a more detailed perspective on data providers and the technological complexities of accessing provided data, we introduce a service that mitigates these complexities by modeling and mapping between heterogeneous data structures, providing integrated access and presenting authority data in an integrated manner. For an initial realization of the service, we have focused on geographical authority data, but expect to integrate other entity types in the near future.

Our contributions are twofold. After analyzing and evaluating data providers and use cases, we identify possibilities to enable access to heterogeneous spatial data through dedicated APIs. Furthermore, we propose integrative access to spatial data by means of a Transformation Service, which is realized on the basis of existing modeling and mapping capabilities, see 4.2.

Authority data in our paper includes data from authority file providers such as the GND, which are curated by their holding institutions, as well as community-based projects like WikiData<sup>8</sup>. With respect to our user-case, both types of data serve the same purpose, identifying entities, and from a technical perspective can be handled similarly in order to achieve the objectives presented in this paper.

---

<sup>6</sup> <https://memorial-archives.international/>

<sup>7</sup> <https://www.oral-history.digital/>

<sup>8</sup> <https://www.wikidata.org/>

## 2 Related Work

The features of authority data include identifiers that can serve to clearly identify entities. In contrast to curated qualitative data from authority file providers, community-based projects, e.g. WikiData or Nominatim<sup>9</sup>, are characterized by user-created data and their immense volume, and can therefore be used to interconnect with information from providers like the GND, to enrich and broaden the data.

Regarding the flexible integration of different authority file providers for spatial data, to the best of our knowledge, no solution is available. A similar, but more general effort has been made to create an overarching authority file across multiple national libraries and archives, resulting in the Virtual International Authority File (VIAF) [Be06]. Other datasets such as the Library of Congress Name Authority Files (LCNAF)<sup>10</sup> serve the same purpose. For these resources, VIAF and LCNAF, an OpenRefine implementation<sup>11</sup> connecting the two datasets has been implemented. The Open Researcher and Contribution Identifier (ORCID), a database for researchers, is another authority file, mainly including data on persons and their publications. However, ORCID has to be separated from VIAF and LCNAF, since researchers themselves can edit, or even remove, their entries and thus the persistence of ORCID data cannot be assured [Pi22, p. 137-138].

On a national level, there have been efforts to integrate authority files from many independent archives and libraries, such as the national network of Italian libraries (Servizio Bibliotecario Nazionale), whose goal is to integrate roughly 12,000 libraries across Italy and their authority files [Ma22]. Another project describes the efforts in the research field technology assessment and its portal for specialists, in which authority files and other data sources are used to identify persons, organization and other data [Ho18]. There, the GND is used in combination with data from ORCID or WikiData in order to identify entities.

Regarding spatial data, the Geobrowser project of the Digital Research Infrastructure for the Arts and Humanities (DARIAH) enables the upload of spatial data in various file formats and visualization on a geographic map<sup>12</sup>. The integration of geographic data in KML, KMZ or CSV formats is available here as well as various options regarding the presentation of the data, e.g. individual layers via file upload (again in KML or KMZ file formats), as well as ArcGIS layers or predefined layers such as historical data for the Roman Empire [Ko16].

The requirement to integrate different authority data providers for geographic data in one infrastructure and thus to enable querying on this heterogeneous data is a new application scenario and will be presented below, after a closer look on data providers.

---

<sup>9</sup> <https://nominatim.org/>

<sup>10</sup> <https://authorities.loc.gov/>

<sup>11</sup> <https://github.com/mcarruthers/LCNAF-Named-Entity-Reconciliation/>

<sup>12</sup> <https://geobrowser.de.dariah.eu/>

### 3 Data Providers

During an initial requirements analysis in the OH.D project, several providers for authority data were identified. These are divided into generic and specialized data sources.

A generic authority file provider that was used for this implementation is the GND entity Geografikum. The access to the data is enabled through the data service entity facts, which comprises a web interface as well as dataset dumps<sup>13</sup> including over 322,000 GND entities. The dataset mostly consists of geographical places, including information like different names of the place and the geographical area code. The metadata included in this dataset is however lacking in some aspects. Many entities contain little to no alternative names, like language or historical variants, which can help to resolve ambiguities. Furthermore, geographic coordinates are only included in roughly 19% of all entities. Some entities contain references to other data sources, such as WikiData that can be used to further enrich the available data, for example to add historical places names.

Other generic authority file sources include GeoNames, a geographical database containing over 12 million entries, which can be downloaded as a data dump. Nominatim, the backbone of OpenStreetMap (OSM), provides an API, that allows access to over 7 billion OSM nodes. In addition to the online API, OSM data can be downloaded and has thus been reused in multiple projects. One prominent example can be found in Photon<sup>14</sup>, an open-source search platform for OSM data.

Apart from the previously mentioned services, which offer generic spatial information, more specialized authority file providers are of particular relevance to the DH and thus represent interesting use-cases for this project. Interviews in OH.D, which comprise reports of contemporary witnesses, often include historic place names. Therefore, general authority data providers, even if they contain alternative place names in their datasets, are not suitable for connecting these historical places to their use in the interviews. In such cases, providers like the Memorial Archives, initiated by the Flossenbürg Concentration Camp Memorial<sup>15</sup>, are needed for more specialized authority data. The Memorial Archives contain data on over 890,000 persons, 4,700 publications, 5,500 places and more. Furthermore, it incorporates references between those different data entries. Regarding spatial data, the type of the place, the time and name are included as well as coordinates. Here, an exemplary use-case for a historical researcher would be using the Memorial Archives in order to get all names for the town Chrastava in the Czech Republic. During the Second World War, a concentration camp near the city was also known by its Polish as well as German name, since the city is close to the Polish and German border. Through specialized data archives such as Memorial Archives, the names of this concentration camp in all three languages are available for further research<sup>16</sup>.

---

<sup>13</sup> <https://data.dnb.de/opendata/>

<sup>14</sup> <https://github.com/komoot/photon/>

<sup>15</sup> <https://www.gedenkstaette-flossenbuerg.de/>

<sup>16</sup> <https://memorial-archives.international/entities/show/56dc6ea9759c022fd48d5286>

The Heterogeneity inherent in these data providers is apparent in different aspects. Direct API access is seldom available, which is why usually data dumps are downloaded and indexed. The dumps themselves come in various formats and require processing in a case-by-case manner.

## 4 Implementation

Authority data providers are heterogeneous in terms of access types (i.e. data downloads or query interfaces), models (i.e. query languages and schemata) and contexts (i.e. universal, national, specific). Our objective is to abstract from these aspects of heterogeneity and to facilitate integrative access to authority data. With regard to harmonizing access to authority data sources, section 4.1 describes the process and technical complexity of creating query APIs based on downloadable data dumps. With harmonized accessibility by means of existing or created APIs, section 4.2 focuses on the idea of overcoming model and context heterogeneity by introduction of the Transformation Service – a mediating component that translates between integrative models and the local models of a dynamic set of data sources.

### 4.1 Data Preparation & Indexing

To create accessible APIs based on available data dumps, microservices are created that process and index the different data sources in Elasticsearch<sup>17</sup> indices. These indices can be searched using the names of places and further filtered with information like country codes and spatial data. Additionally, individual entities can be requested by their respective identifier. The APIs use POST and GET Requests, with JSON request bodies and are automatically built and deployed as Docker images using continuous integration and deployment pipelines. First, the data dumps need to be processed and indexed. To keep data up to date they are reindexed in fixed intervals.

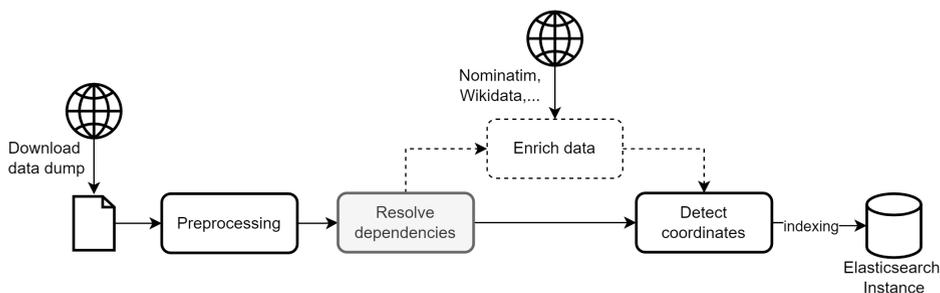


Fig. 1: Procedure of getting raw data, processing and indexing it to a Elasticsearch instance.

<sup>17</sup> <https://www.elastic.co/elasticsearch/>

Fig. 1 shows the process from downloading data sources to indexing them with Elasticsearch. The step *Resolve dependencies* is not necessary for all data sources. Also, the step *Enrich data* is not implemented yet and is further discussed in Section 5. So far the database dumps from the authority file providers GND Geografikum and GeoNames have been considered and indexed to an Elasticsearch instance, a next step will be to integrate the crawled Memorial Archives geographic data. Elasticsearch has been chosen due to other software projects that are connected or will be linked to this service in the future. It allows metadata to be queried in different ways, for example by filtering based on spatial information, such as coordinates or country codes.

For illustration purposes, the indexing of the Geografikum data dump is further described. The files containing the spatial data are available in several formats. Due to the use of Elasticsearch the format JSON-LD was chosen, since it is easy to integrate. The JSON-LD file contains entries with three different types of identifiers. One describes a GND entity (e.g. <https://d-nb.info/gnd/4004391-5>), which references entries with other types of identifiers, if available. The second type of identifier (e.g. <https://d-nb.info/gnd/4004391-5/about/>) carries further information about the entity. For almost one fifth of the GND entities, a third type of identifier (e.g. `_:node1gf2tc0d5x21379656`) is referenced, which points to an entry in the file containing the coordinates of the location. Before indexing, the dependencies between the entries in the JSON-LD file need to be resolved and integrated into one entity. Lastly, the coordinates are defined as so called Geopoints to allow querying and filtering by distance or location using Elasticsearch.

## 4.2 Data and Service Consolidation

Based on applied schemata and technical contexts of providers, APIs expect requests in specific forms (i.e. input models) and provide responses in terms of output models. In consequence, services with a need for integrated access to multiple data sources such as OH.D need to implement access to heterogeneous interfaces, input and output models. Emerging from the modeling capabilities of the DARIAH-DE Data Modeling Environment (DME)<sup>18</sup> [GH16, HG21] we propose the concept of a generic Transformation Service. Among other roles, the service acts as intermediary to online interfaces – mediating between integrative data demands and the accessible, yet heterogeneous sources of relevant data. Integrated query and result models are tailored to individual needs – here the OH.D portal and its technical setting.

Fig. 2 outlines the main functionality of the service with particular focus on the idea of interface mediation: Users of the OH.D portal formulate queries in terms of a defined query model and submit them to an API provided by the service. By applying mappings between integrated and local input models, queries can be translated and executed against applicable

<sup>18</sup> <https://de.dariah.eu/en/dme/>

data sources. Returned results are collected and sent to the user in terms of an integrated result model – again by applying relevant mappings.

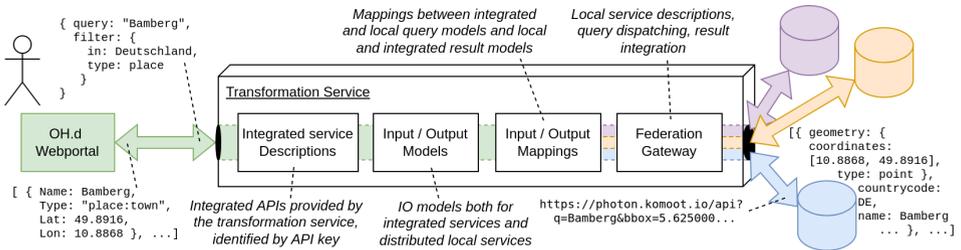


Fig. 2: Transformation service

In order to address initial use cases of OH.D, multiple APIs have been configured to each query one primary data source, whose results are enriched by queries to secondary sources. Primary and secondary geographical data sources can be configured individually per interview collection. All APIs oriented towards the OH.D portal share the same input and output models, despite binding to different data sources.

## 5 Discussion and Outlook

The advancements presented in this paper are embedded in the infrastructure of the language- and text-based infrastructure NFDI<sup>19</sup> project Text+ and can thus be re-used for future application scenarios. The integrated access to geographic information will enhance the capabilities of the Transformation Service (detailed in 4.2), as intermediary between requests, interfaces and data sources.

Still, the integration of data providers is affected by different file formats and access modalities. Thus, a procedural approach is necessary in order to handle the integration of additional data providers to enable comprehensive access.

To further improve the metadata, data providers described in Section 3 can be used. First, the number of geographic locations can be increased by utilizing more data providers like Nominatim and the Memorial Archives. Next, the data quality can be enriched by incorporating more historical place names or adding missing geographic information, like coordinates or country codes. Moreover, other types of entities such as persons or historical events should be considered in the future. Additional data dumps from GND and further sources of authority data can be used for this to index and search the data in a similar way to geographical data addressed in this work.

Our system is currently used as part of the OH.D project in order to identify spatial information and enrich metadata of interview transcripts. In doing so, feedback from researchers is raised and influences further development.

<sup>19</sup> <https://www.nfdi.de/>

## Bibliography

- [Be06] Bennett, Rick; Hengel-Dittrich, Christina; O'Neill, Edward T; Tillett, Barbara B: Vial (virtual international authority file): Linking die deutsche bibliothek and library of congress name authority files. In: World library and information congress: 72nd IFLA general conference and council. 2006.
- [GH16] Gradl, Tobias; Henrich, Andreas: Data Integration for the Arts and Humanities : A Language Theoretical Concept. In: Research and Advanced Technology for Digital Libraries 20th International Conference on Theory and Practice of Digital Libraries, TPDL 2016, Hannover, Germany, September 5–9, 2016, Proceedings, pp. 281–293. Springer International Publishing, Cham, Switzerland, 2016.
- [HG21] Henrich, Andreas; Gradl, Tobias: Integration von Forschungsdaten : Wie können Forschungsinfrastrukturen helfen? In: Innovation in der Bauwirtschaft, pp. 749–786. De Gruyter, Berlin, Boston, 2021.
- [Ho18] Hommrich, Dirk; Pasucha, Beate; Razum, Matthias; Riehm, Ulrich: Normdaten und Datenanreicherung beim Fachportal openTA. *Bibliotheksdienst*, 52(3-4):248–265, 2018.
- [Ko16] Kollatz, Thomas: Raum-zeit-analysen mit geo-browser und datasheet-editor. *Bibliothek Forschung und Praxis*, 40(2):229–233, 2016.
- [Ma22] Mataloni, Maria Cristina: Integrated Search System: evolving the authority files. *Bibliographic Control in the Digital Ecosystem*, 7:335–346, 2022.
- [Pi22] Piazzini, Tessa: Bibliographic control and institutional repositories: welcome to the jungle. *Bibliographic Control in the Digital Ecosystem*, 7:132–142, 2022.

# Geo Engine: Workflow-backed Geo Data Portals

Christian Beilschmidt<sup>1</sup> Johannes Drönner<sup>1</sup> Michael Mattig<sup>1</sup> Philip Schweitzer<sup>1</sup> Bernhard Seeger<sup>1,2</sup>

**Abstract:** Geo data portals play a key role in the distribution and exploitation of domain-specific geo data. While such portals are highly specialized, they share a number of common requirements that span from data access and processing to UI components. Geo Engine is able to provide all the necessary parts for portal building. We demonstrate this on a real data portal we built for the dragonfly community. In addition, we show its general architecture and outline future improvements.

**Keywords:** Geo processing; Data portals

## 1 Introduction

In recent years, many tailor-made geo data portals have arisen to provide specific data and services to a domain-specific user group. The primary goal is to provide powerful geo-temporal insight as simply as possible for a few dedicated use cases. On the other hand, geographic information systems (GIS) have matured over the last two decades to address the needs of processing Big Data regarding volume, variety, and velocity. However, there is a high demand for systems providing powerful processing and the ability to create customized portals to empower non-technical users to take full advantage of open and FAIR data [Wi16]. So far, current portals are not capable of dealing with Big Data.

The foundation of generating customized geo data portals are building blocks for accessing large raster and vector data, geo-temporal workflow processing and analysis, and flexible data access by standard-compliant interfaces. These blocks are already integral components of Geo Engine, a novel service platform that has already served as the basis for various portals. Geo Engine is the successor of the VAT system (cf. Sect. 5), which already powered GFBio's [Di14] data portal until 2021. We took the lessons learned from our five-year development of VAT and designed Geo Engine as an entirely new system that overcomes previous limitations. Geo Engine offers new functionality to empower (data) scientists dealing with large and heterogeneous datasets via different semantically equivalent interfaces (e.g., a Python interface for programming enthusiasts and a no-code interface for less technically experienced users). Among the many novel features of Geo Engine is its abstraction for data access and its consistent temporal approach to treating every data item as a time series.

---

<sup>1</sup> Geo Engine GmbH, Am Kornacker 68, 35041 Marburg, Germany {firstname.lastname}@geoengine.de

<sup>2</sup> University of Marburg, Dept. of Mathematics and Computer Science, Hans-Meerwein-Str., 35032 Marburg, Germany seeger@mathematik.uni-marburg.de

This makes Geo Engine a unique system for geo-spatial time series processing. Geo Engine is also a progressive system [Be19, Ho20] supporting the early delivery of approximate results to users to support an interactive and exploratory way of working on Big Data. In addition, Geo Engine offers the building blocks for customized portals such that only a thin mashup layer is required to meet users' specific demands.

This paper is structured as follows: In Sect. 2, we present Geo Engine's architecture and describe its data and processing model, as well as abstractions and components. In Sect. 3, we outline the requirements of data portals and how Geo Engine tackles them by providing suitable building blocks. In Sect. 4, we showcase a demo portal that is built on top of Geo Engine. In Sect. 5, we present related work. Finally, in Sect. 6, we give a brief summary and point out future directions of Geo Engine.

## 2 Geo Engine: Architecture Overview

Geo Engine consists of a backend<sup>3</sup> and two frontends: *geoengine-ui*<sup>4</sup> for Web and *geoengine-python*<sup>5</sup> (Fig. 1). The backend handles data access, data management and query execution. It also provides APIs for the frontends and third-party applications. OGC<sup>6</sup>-compliant interfaces, e.g. Web Map Service (WMS), Web Coverage Service (WCS), and Web Feature Service (WFS), allow access to data layers and computed layers derived on-the-fly at runtime. This makes Geo Engine compatible with most other geo software. The remaining functionality is available through a custom RESTful Web API with an OpenAPI<sup>7</sup> specification. We deploy Geo Engine using OCI<sup>8</sup> containers (e.g. Docker) where the backend and the selected frontend run in separate containers.

The backend of Geo Engine is written in Rust, a system language that overcomes many of the deficiencies of C and C++. It consists of three modules: data types, operators, and services. *Data types* contains the primitives for vector and raster data as well as basic operations and spatial projections. *Operators* contains the spatio-temporal query execution engine and the implementation of operators. *Services* contains the data management, i.e. adding, updating and removing artifacts such as datasets, workflows and projects, and Web APIs on top of this functionality. Optionally, it also handles user management, authentication via OpenID Connect [Sa14] single sign-on (SSO) providers, and authorization, which allows restricting access to resources such as data and workflows to certain users and groups.

The main output of Geo Engine are *layers* of spatio-temporal data: either feature collections or raster images. *Workflows* specify the processing of the *layers* as a graph of operators. All

---

<sup>3</sup> [github.com/geo-engine/geoengine](https://github.com/geo-engine/geoengine)

<sup>4</sup> [github.com/geo-engine/geoengine-ui](https://github.com/geo-engine/geoengine-ui)

<sup>5</sup> [github.com/geo-engine/geoengine-python](https://github.com/geo-engine/geoengine-python)

<sup>6</sup> [www.opengeospatial.org](http://www.opengeospatial.org)

<sup>7</sup> [www.openapis.org](http://www.openapis.org)

<sup>8</sup> [www.opencontainers.org](http://www.opencontainers.org)

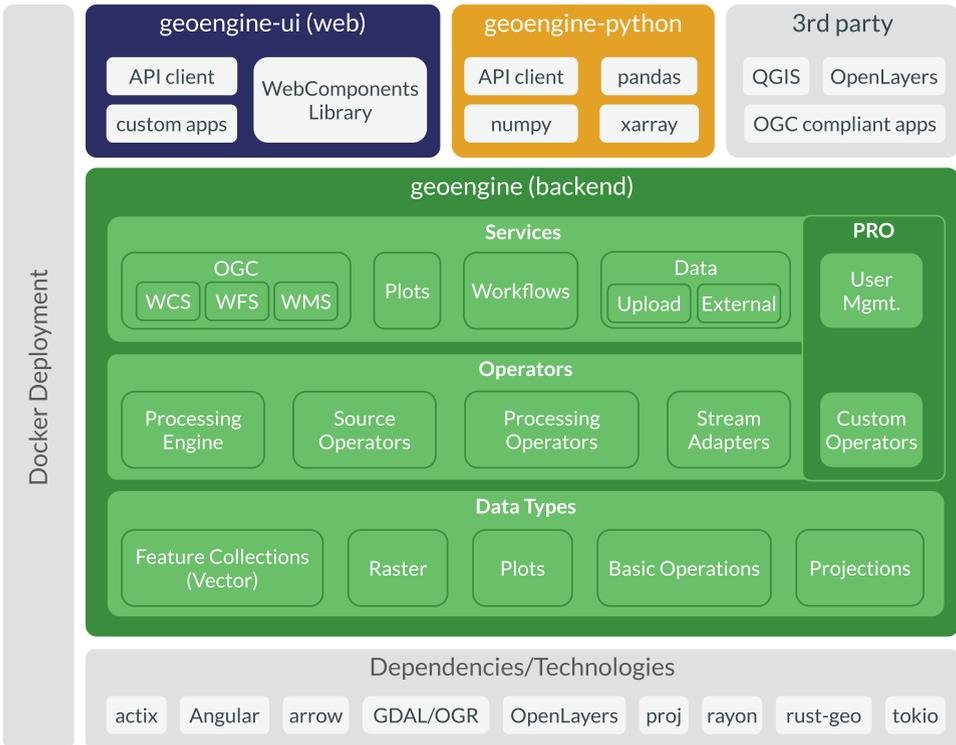


Fig. 1: A high-level overview of Geo Engine's architecture and its main components. The features in PRO are not open source but source-available. They are also freely available for non-commercial users and projects.

workflows consist of input operators and optionally processing operators. The two most important input operators are the *GdalSource* and the *OgrSource* that handle raster and vector data loading, respectively. They use the omnipresent GDAL library [Ro22] in order to support a variety of geospatial data formats as input. Processing operators either transform one piece of data into another, e.g. by filtering, or combine multiple inputs into a new output, e.g. attaching raster values to a point collection.

All data in Geo Engine is spatio-temporal and homogeneous. This means in a single raster all cells have the same size and the same data type. In contrast to data cubes, however, different rasters can still have other data types and sizes and will only be harmonized when it is necessary for a computation. In a feature collection, all features are of the same type, e.g. MultiPoint (cf. simple feature model [Op10]). This is necessary to define meaningful operations, e.g., filtering a collection's points based on them being contained in another collection's polygons.

Everything also has a temporal validity, defined as a half-open time interval [start, end). In a raster, all cells have the same temporal validity. In a feature collection, each feature has its own temporal validity. If a feature has multiple geometries, e.g. points in a MultiPoint, all geometries have the same temporal validity.

In order to enable the processing of datasets larger than the available main memory, all computations are performed on streams of chunks (vectors) and tiles (rasters) of a fixed size. A stream is a Rust data type that allows asynchronously producing and consuming data. Input operators produce chunks when they read the data and operators can *await* these chunks to be ready. In turn, other operators that use these outputs can again await them. This allows Geo Engine to interleave data loading and processing because operations do not block. The actual work is performed in a thread pool with a fixed number of workers, which is chosen with respect to the number of CPU cores.

Raster tile streams are produced as *time first*, and *space second*. Starting with the first image in a raster time series, we produce the tiles starting from the top-left and increasing right and then down. Then, we continue with the image of the next time step in the same way. First of all, it is crucial to fix the tile order, such that consuming (parent) operators can rely on these order guarantees. Moreover, from our experience, this particular order is suitable for most common spatial operations on time series. For different requirements, e.g., temporal aggregations of single pixels, Geo Engine employs adapters that break the problem down into a set of subqueries that contain exactly one tile to generate a time-first stream.

Feature collections are split up by a fixed size limit such that chunks are of roughly the same size. Operators in turn have to produce new output chunks/tiles from input chunks. This sometimes requires reading the same piece of data multiple times, e.g. for a join or a convolution. This can be mitigated by employing an LRU cache to alleviate this problem of redundant computations.

For rasters, all processing is performed on a global grid with a fixed origin and a fixed tile size (e.g.  $512 \times 512$  pixels). For a given query bounding box (BBox), we compute all the tiles that intersect this BBox. The major advantage of uniform tiling is that it allows us to easily combine multiple rasters, as the tiles are always aligned. It also allows easier re-use of cached results, because elements can be taken as they are and not be stitched together.

Geo Engine can access internal and external data. Our approach is to identify loadable data by an ID. An input operator obtains this ID as a parameter and resolves the necessary loading information using a metadata provider. This information is, for instance, the file name, the location, and the used spatial projection. Here, users can also specify regular time series by file names with date templates or a list of irregular time steps of a time series. Internal data are stored as *datasets* in a database. User can create their own datasets and share them with other users. External data is provided by *Data Providers*.

Data Providers allow (1) to browse and (2) to access data that is not managed by Geo Engine itself. In contrast to internal datasets, external data is referenced by an *external data ID* that

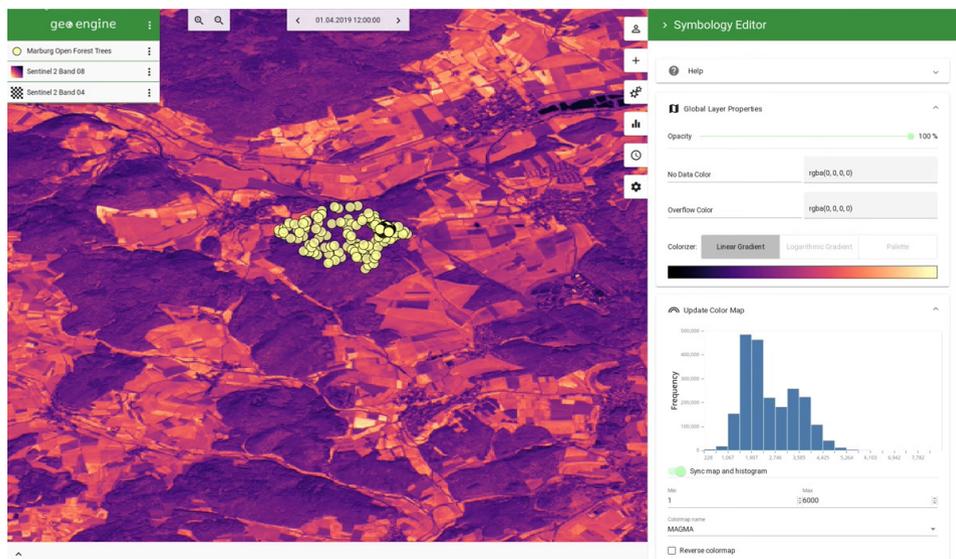


Fig. 2: A screenshot of Geo Engine's GIS UI.

combines a *provider ID* with a *layer ID*. When an input operator gets an external data ID, it uses the data provider to resolve the necessary loading information for the layer ID. In contrast to local datasets, external data cannot be edited or deleted and the available data may change over time. Some examples of external data providers are generic WCS and STAC services, and more specialized ones like the GEO BON EBV Data Portal<sup>9</sup> and NFDI Core Storage<sup>10</sup>. To browse all available data, Geo Engine exposes internal and external data in a uniform layer collection API.

Our Web frontend *geoengine-ui* consists of three parts: a core library, a GIS application, and multiple apps and dashboards. The core provides a client implementation for the backend API services, e.g. for managing layers, and building block components like the map, plots, and operator dialogs. These components are all interconnected via application-wide states and services within a reactive application architecture. For instance, a data table and a map layer would relate to the same layer object that is managed by the core library. The GIS application (Fig. 2) offers the full functionality of Geo Engine, which is targeted at expert users. They can work with multiple layers, apply operators and review workflow graphs. All views (map, plots, data table) are synchronized and automatically adjust to the selected time and map extent. For instance, when a user pans to a different spatial area, a histogram plot would be recomputed to reflect the visible data on the screen. The dashboards are much simpler applications that focus on a concrete use case and only require access to a few

<sup>9</sup> portal.geobon.org

<sup>10</sup> www.nfdi4biodiversity.org

selected inputs. This allows for building easy-to-use domain-specific dashboards that still are able to leverage the full power of the Geo Engine data access, operator engine, and UI components.

### 3 Building Data Portals using Geo Engine

Data portals are highly specific to their target audience, but most of them share the same fundamental requirements. From our experience, the following list captures the most important properties of a modern geo data portal:

- R1** Flexible data access to different data types and formats
- R2** Combining local and remote data sources
- R3** Basic layers and derived layers using GIS operations on available data
- R4** A map as a central dashboard component
- R5** A web-based user interface and reusable components
- R6** User interactions, e.g. panning, zooming or selecting data subsets
- R7** Time functionality for working with time series
- R8** Multi-views, e.g. data tables and plots
- R9** Access control, i.e. ensuring data privacy and having a multi-user system
- R10** Administration tools for defining and managing the portal
- R11** Response times that allow working interactively

The combination of rich data access (R1), flexible layer definitions that leverage workflows, basic GIS operators, and a UI component library makes it easy and efficient to build interactive geo data portals using Geo Engine. The data can either be added as internal datasets or accessed externally using a data provider, for instance, accessing a project database (R2). For displaying data as layers on the map, data is either used as it is or processed using advanced workflows (R3). In both cases, users can group the data as needed and uniformly browse them via our layer collection API. Then, the portal offers different display options, e.g., a simple list or cascaded dropdown lists for hierarchies (R6). Furthermore, the portal is able to show multiple datasets at once and, e.g., to enrich project data with other public data. As Geo Engine supports time as an integral dimension, it is easy to realize time sliders (R7). Moreover, in addition to the map, a portal can provide different kinds of plots and tabular views of the data (R8). If required, it can offer users to

dynamically input their data, e.g., by drawing areas of interest, which act as new datasets that are read-to-use for processing in the portal.

Data is, by default, only processed with respect to the current map resolution, i.e. the number of pixels and the zoom level that is currently visible to the user. This allows Geo Engine to perform the calculations on overviews and avoids processing all the data at their finest level of detail. This leads to high interactivity (R11), even supporting a large number of operators and computing steps. If computations take too long for reasonable user interaction, administrators can also save the result of a workflow as a new dataset upfront and avoid recomputations.

Setting up a data portal currently consists of creating a new project that builds upon the core library of the `geoengine-ui` project. Then, developers can combine existing components into a domain-specific dashboard that can be enriched with custom texts and explanations. Additionally, they can define specific color schemes and styles, and add project-specific icons and logos. After setting up the application, one can set up datasets, layers, and other config items in the backend (R10). Datasets and layers are defined as JSON files and either loaded during the start of the backend or added later via REST. The layer IDs can be stored in the frontend's config for their retrieval at runtime. Finally, the dashboard is packaged as a container and deployed alongside a backend instance. For each user of the data portal, Geo Engine will create an anonymous user account on the fly and the user is logged into the system automatically. This allows users to get their own private session while avoiding a registration. However, it is also possible to add a global password or individual user accounts to an instance to allow access only to a specific target audience (R9).

Some of the main UI building blocks are the map, the layer list, the data table, plots, legends, and colorizers (R8). The map is typically the central point of attention in a geo data portal (R4). It overlays multiple layers of raster and vector data. The rasters are styled in the backend using a colorizer. The colorizer is predefined for each layer but can be changed by the user in an editor if it is integrated into the portal. Vectors are styled in the frontend itself and can also be customized, e.g., by varying the size of points depending on the value of an attribute. The plots are computed in the backend, but rendered in the frontend using the Vega plot grammar [Sa17]. This allows for a good visual quality and plot interactions but requires little computing resources. While in our GIS the plots are always linked to the map and data table with respect to time and space, portal creators are free to set time and space for maps and plots independently. Thus, it is, e.g., possible to show a plot for a whole year, while navigating through time month-by-month on the map.

Geo Engine's core UI library is based on Angular<sup>11</sup> components, which pose a form of Web Component<sup>12</sup> implementation (R5). Since Web Components are not yet fully standardized and thus have varying implementations, project dashboards currently need to be Angular

---

<sup>11</sup> [www.angular.io](http://www.angular.io)

<sup>12</sup> [www.webcomponents.org](http://www.webcomponents.org)

projects as well. Since Angular and its Material Design<sup>13</sup> look-and-feel are widely used on the Web, they are familiar to users. In the future, it is likely that these components can more easily be used within different Web frameworks.

In addition to UI building blocks, for enthusiasts and advanced users, portal providers can use the `geoengine-python` library to allow users extended capabilities. Users can then get the portal data as geo data frames and `xarrays` [HH17] into their Jupyter Notebooks. There they can perform extended analyses and combinations with their own datasets.

## 4 A Use Case: Dragonfly Portal

As part of the NFDI4Biodiversity project, and in cooperation with the society of German-speaking odonatologists (GdO e.V)<sup>14</sup> we built a demo of a dragonfly geo portal for the GdOnline 2022 conference [GdOe22]. The portal allows visualizing occurrence data of dragonflies<sup>15</sup> for all of Germany. It offers some basic analysis functions, e.g., correlating occurrences with monthly temperature aggregates from remote sensing models (ECMWF ERA-5 Land [MS19]) or land cover based on Sentinel-2 data [Ri21]. It is meant as a collective hub of information about dragonflies (descriptions, pictures) and a contact point for people interested in dragonflies. Thus, it is a means to increase the visibility of the valuable work of volunteers who collect the data.

The portal is divided into two parts (Fig. 3). On the left-hand side, the data selection and plots are placed. On the right-hand side, we see the map. The user can select a dragonfly species by name and add an additional environmental layer, e.g. temperature or land use. Optionally, they can also activate the sampling frequency layer which gives some context to the absolute occurrence number shown in the observation points. This sampling frequency is pre-computed by a Geo Engine workflow of all dragonfly occurrences within a certain time period and the application of a grid-based rasterization that counts the number of observations per square.

The time selector on the top allows selecting the year. All observations within this year will be shown on the map as an aggregated number. This is internally done by applying a workflow that projects the temporal validity of the occurrences to full months. Doing this aligns the occurrences with the temporal selection as well as the temporal validity of the environmental layers. To avoid clutter and yield a better visualization, the points are clustered using the *VisualPointClustering* [Be17b] operator to present an overlap-free representation. The second time slider near the bottom allows selecting the month within the year. This slider is only relevant to the histogram plot that is optionally calculated when clicking a button at the bottom. The plot visualizes the correlation between the occurrences and the selected environmental variable within the chosen month.

---

<sup>13</sup> [www.material.io](http://www.material.io)

<sup>14</sup> [www.libellula.org](http://www.libellula.org)

<sup>15</sup> Demonstration data from AK Libellen NRW (2020), [www.ak-libellen-nrw.de](http://www.ak-libellen-nrw.de)

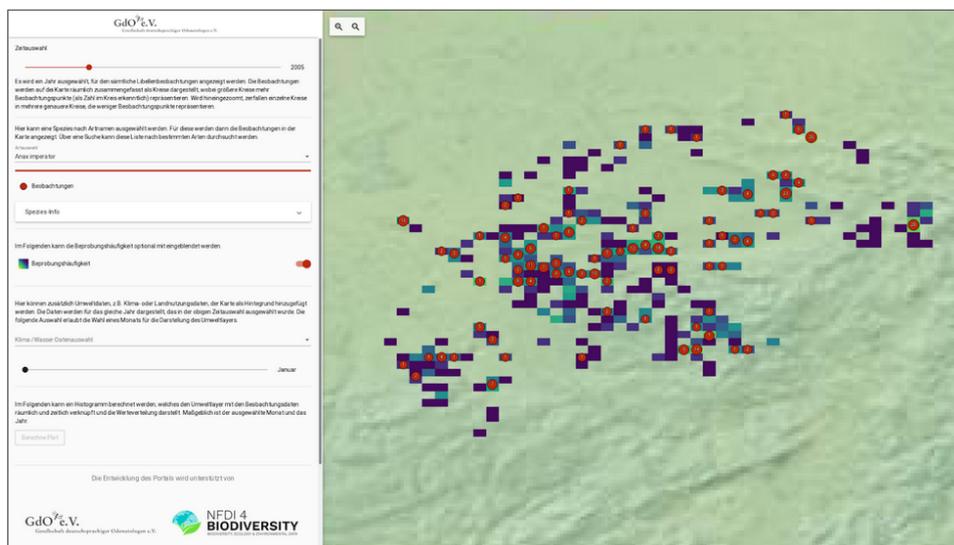


Fig. 3: A screenshot of the GdO e.V. dragonfly portal demonstrator.

The dragonfly occurrence points use a number of Geo Engine features. The data is loaded from a GeoPackage database via Geo Engine's *OgrSource*. The filtering of points by species and aggregation over time is performed by the workflow processing engine. The selected time is used in the query rectangle of the query and allows reusing a fixed set of workflows for different points in time. In addition to the map view, the layer selection and legend displays are used from the core library. The plots use Geo Engine workflows to combine the occurrence points with the environmental raster data. This raster data is loaded from a set of Cloud-Optimized GeoTiffs (COG) via the *GdalSource*. The plots' data itself is calculated in the backend and displayed by the plot component from the core library.

The portal demonstrator poses an easy-to-use GIS subset within a domain-specific data portal. The portal providers can predefine a fixed set of data and derived data by registering workflows in the Geo Engine for future re-use. Note, that it is possible to use either Jupyter notebooks or the Geo Engine GIS application to formulate these workflows. In conclusion, the demonstrator is a successful example of leveraging Geo Engine's existing functionality instead of developing a tailor-fit data portal from scratch.

## 5 Related Work

Geo Engine is the successor of the VAT System [Au15b, Au15a, Be17a] that was developed as part of the GFBio project [Di14]. The original purpose of VAT was to make GFBio data more easily accessible, but over time it developed into a more full-fledged geo data analysis

platform. In comparison to VAT, Geo Engine offers a great number of improvements, of which we highlight the three most important ones. While VAT could only work on single rasters, Geo Engine is able to process arbitrary raster time series. Both for raster and vector data, Geo Engine is no longer limited by the amount of available main memory. While VAT required implementing special operators to access external data, Geo Engine introduces the data providers that only require creating the loading information and reusing the basic input operators.

A notable type of geoprocessing software that tackles similar problems as Geo Engine are data cubes. One representative is the Open Data Cube (ODC) [Ki18]. In contrast to Geo Engine, ODC harmonizes all data upfront while building the  $n$ -dimensional data cube. This makes it far more inflexible as the requirements, e.g., for the resolution and the included datasets must be known up-front, or the data cube has to be rebuilt. It is limited by the available main memory and does not support vector data in the processing in the same fashion as Geo Engine.

Google Earth Engine<sup>16</sup>, as a representative of cloud GIS services, [Go17] is a popular tool for analyzing earth observation data. It provides a range of datasets that are ready to use and a variety of processing tools. In contrast to Geo Engine, Google Earth Engine is not Open Source and cannot be hosted on-premise. All data that is to be analyzed, has to be uploaded to Google. Also, the Google Earth Engine does not support temporal processing as a core feature of its language but rather has to be performed manually in its supported scripting language.

Carto<sup>17</sup> and Mapbox<sup>18</sup> are two providers that specialize in the creation of maps. Here, the focus is not on processing, but on designing good-looking and highly informative maps. In contrast to Geo Engine, there is no focus on processing pipelines and supplying GIS-like interfaces for generic geoprocessing.

GeoNode<sup>19</sup> is a data management platform that builds upon GeoServer [Ia17]. It allows non-expert users to create interactive maps and to publish data for external tools. As its focus is on data management and sharing, it lacks a flexible toolbox to process the data beyond simple filter mechanisms. In contrast to Geo Engine, analysis is done with external GIS tools, e.g., QGIS<sup>20</sup>, rather than having workflows within GeoNode itself.

In the context of biodiversity data, GBIF hosted portals<sup>21</sup> are an easy way of creating custom geo portals. The portals make use of the existing GBIF infrastructure which alleviates the work and costs of hosting. However, the portals are limited to the data that is already

---

<sup>16</sup> [earthengine.google.com](http://earthengine.google.com)

<sup>17</sup> [www.carto.com](http://www.carto.com)

<sup>18</sup> [www.mapbox.com](http://www.mapbox.com)

<sup>19</sup> [www.geonode.org](http://www.geonode.org)

<sup>20</sup> [www.qgis.org](http://www.qgis.org)

<sup>21</sup> [www.gbif.org/hosted-portals](http://www.gbif.org/hosted-portals)

available on GBIF. Also, portals can only be created after a successful application and review by GBIF.

The Atlas of Living Australia<sup>22</sup> (ALA) offers open access to Australia's biodiversity data. It offers a rich set of feature modules for discovering and visualizing geo data, which in part uses existing geo software like GeoServer. The ALA software can be used for custom geo portals as well. In contrast to Geo Engine, it does not allow for custom interactive analysis. Also, it cannot be readily installed by anyone, but only on an individual per-request basis.

## 6 Summary and outlook

In this paper, we presented Geo Engine's architecture and its usage for powering geo data portals. For this, we outlined general requirements for modern data portals and how Geo Engine fulfills them. Moreover, we presented a use case of a Geo Engine data portal presenting dragonflies. Finally, we compared Geo Engine to related systems.

In the future, we will focus on improving the creation and administration of geo portals with Geo Engine. We are working on a portal builder that eliminates the need to write custom code in our `geoengine-ui` repository (which took the majority of the time in building the dragonfly portal). Instead, it will be possible to create dashboards declaratively in the Geo Engine UI. This will also make it easier to host multiple geo portals using a single Geo Engine instance. Currently, a single backend instance can already serve multiple frontends, but each portal frontend has to be deployed individually. We will also provide an administration UI for managing existing portals. This will reduce the required technical knowledge for operating portals. Furthermore, we are going to develop more geo portals ourselves as blueprints. This will help the adoption of our technology for portal building and facilitate community building.

For improving Geo Engine as a platform, we will create more functionality such as new operators to support a greater variety of use cases. In particular, we will develop means for creating machine learning models that are fed by Geo Engine's data pipeline. Then, experts can learn a model using Geo Engine's full-fledged capabilities. In data portals, one can use these models and apply them simply as another operator of a Geo Engine workflow.

## Acknowledgments

This work was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant number O3EUPHE069.

---

<sup>22</sup> [www.ala.org.au](http://www.ala.org.au)

## Bibliography

- [Au15a] Authmann, C.; Beilschmidt, C.; Drönner, J.; Mattig, M.; Seeger, B.: Rethinking spatial processing in data-intensive science. In: *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft für Informatik (GI)*. volume 242, 2015.
- [Au15b] Authmann, Christian; Beilschmidt, Christian; Drönner, Johannes; Mattig, Michael; Seeger, Bernhard: VAT: A System for Visualizing, Analyzing and Transforming Spatial Data in Science. *Datenbank-Spektrum*, 15(3):175–184, 2015.
- [Be17a] Beilschmidt, Christian; Drönner, Johannes; Mattig, Michael; Seeger, Bernhard: VAT: A System for Data-Driven Biodiversity Research. 20th International Conference on Extending Database Technology (EDBT), 2017.
- [Be17b] Beilschmidt, Christian; Fober, Thomas; Mattig, Michael; Seeger, Bernhard: A Linear-Time Algorithm for the Aggregation and Visualization of Big Spatial Point Data. In: *SIGSPATIAL '17: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, New York, NY, USA, pp. 73:1–73:4, 2017.
- [Be19] Berg, Lukas; Ziegler, Tobias; Binnig, Carsten; Röhm, Uwe: ProgressiveDB - Progressive data analytics as a middleware. In: *Proceedings of the VLDB Endowment*. volume 12, 2019.
- [Di14] Diepenbroek, Michael; Glöckner, Frank Oliver; Grobe, Peter; Güntsch, Anton; Huber, Robert; König-Ries, Birgitta; Kostadinov, Ivaylo; Nieschulze, Jens; Seeger, Bernhard; Tolksdorf, Robert; Triebel, Dagmar: Towards an Integrated Biodiversity and Ecological Research Data Management and Archiving Platform: The German Federation for the Curation of Biological Data (GFBio). In: *GI-Jahrestagung*. volume 232 of LNI, GI, Bonn, Germany, pp. 1711–1721, 2014.
- [GdOe22] Gesellschaft deutschsprachiger Odonatologen e.V., Heidelberg: GdOnline 2022. In: 2. Digitalkonferenz der Gesellschaft deutschsprachiger Odonatologen (GdO e.V.), 18.-19. März 2022. 2022.
- [Go17] Gorelick, Noel; Hancher, Matt; Dixon, Mike; Ilyushchenko, Simon; Thau, David; Moore, Rebecca: Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 2017.
- [HH17] Hoyer, Stephan; Hamman, Joe: xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1), 2017.
- [Ho20] Holanda, Pedro; Raasveldt, Mark; Manegold, Stefan; Mühleisen, Hannes: Progressive indexes: Indexing for interactive data analysis. In: *Proceedings of the VLDB Endowment*. volume 12, 2020.
- [Ia17] Iacovella, Stefano: *GeoServer Beginner's Guide: Share Geospatial Data using Open Source Standards*. Packt Publishing Ltd, 2017.
- [Ki18] Killough, Brian: Overview of the open data cube initiative. In: *International Geoscience and Remote Sensing Symposium (IGARSS)*. volume 2018-July, 2018.
- [MS19] Muñoz Sabater, J.: ERA5-Land Monthly Averaged Data From 1981 to Present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2019.

- [Op10] Open Geospatial Consortium: OpenGIS Implementation Standard for Geographic Information - Simple Feature Access. OpenGIS Project Document, 2010.
- [Ri21] Riembauer, Guido; Weinmann, Anika; Xu, Shaojuan; Eichfuss, Silas; Eberz, Charlotte; Neteler, Markus: Germany-wide Sentinel-2 based land cover classification and change detection for settlement and infrastructure monitoring. In: Proceedings of the 2021 conference on Big Data from Space. pp. 53–56, 2021.
- [Ro22] Rouault, Even; Warmerdam, Frank; Schwehr, Kurt; Kiselev, Andrey; Butler, Howard; Loskot, Mateusz; Szekeres, Tamas; Tourigny, Etienne; Landa, Martin; Miara, Idan; Elliston, Ben; Kumar, Chaitanya; Plesea, Lucian; Morissette, Daniel; Jolma, Ari; Dawson, Nyall; GDAL, 7 2022.
- [Sa14] Sakimura, Natsuhiko; Bradley, John; Jones, Mike; De Medeiros, Breno; Mortimore, Chuck: OpenID Connect Core 1.0. The OpenID Foundation, p. S3, 2014.
- [Sa17] Satyanarayan, Arvind; Moritz, Dominik; Wongsuphasawat, Kanit; Heer, Jeffrey: Vega-Lite: A Grammar of Interactive Graphics. IEEE Transactions on Visualization and Computer Graphics, 23(1), 2017.
- [Wi16] Wilkinson, Mark D.; Dumontier, Michel; Aalbersberg, IJsbrand Jan; Appleton, Gabrielle; Axton, Myles; Baak, Arie; Blomberg, Niklas; Boiten, Jan-Willem; da Silva Santos, Luiz Bonino; Bourne, Philip E.; Bouwman, Jildau; Brookes, Anthony J.; Clark, Tim; Crosas, Mercè; Dillo, Ingrid; Dumon, Olivier; Edmunds, Scott; Evelo, Chris T.; Finkers, Richard; Gonzalez-Beltran, Alejandra; Gray, Alasdair J.G.; Groth, Paul; Goble, Carole; Grethe, Jeffrey S.; Heringa, Jaap; 't Hoen, Peter A.C; Hooft, Rob; Kuhn, Tobias; Kok, Ruben; Kok, Joost; Lusher, Scott J.; Martone, Maryann E.; Mons, Albert; Packer, Abel L.; Persson, Bengt; Rocca-Serra, Philippe; Roos, Marco; van Schaik, Rene; Sansone, Susanna-Assunta; Schultes, Erik; Sengstag, Thierry; Slater, Ted; Strawn, George; Swertz, Morris A.; Thompson, Mark; van der Lei, Johan; van Mulligen, Erik; Velterop, Jan; Waagmeester, Andra; Wittenburg, Peter; Wolstencroft, Katherine; Zhao, Jun; Mons, Barend: The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data, 3(1):160018, 12 2016.



# Semantic Search for Biological Datasets: A Usability Study on Modes of Querying and Explaining Search Results

Felicitas Löffler,<sup>1</sup> Fateme Shafiei,<sup>2</sup> René Witte,<sup>3</sup> Birgitta König-Ries,<sup>4</sup> Friederike Klan<sup>5</sup>

**Abstract:** Dataset discovery is a frequent task in daily research practice, yet studies are missing that explore the usability of user interfaces (UI) in data portals. In particular, very few user studies exist that analyze whether particular elements in the user interface are useful for search tasks. We aim to address those needs for more specific usability evaluations in dataset search. In this work, we present a flexible semantic search over biological datasets with two user interfaces. The search result contains semantically related terms, such as synonyms or more specific terms, obtained from domain ontologies. We evaluated the system in a user study with 20 scholars. We focused on two components, the query input to explore a search in categories (entity types) in comparison to a single input field, and we analyzed textual highlightings in the returned datasets to study whether users are distracted by semantic information such as URIs. Our results show that users prefer interfaces with a single input field for search tasks they are not familiar with, and that users appreciate explanations with terminologies and URIs.

**Keywords:** Dataset Search; Semantic Search; Biodiversity; Life Sciences; User Interface; Usability

## 1 Introduction

Data-intensive research increasingly requires scientists to retrieve datasets in data portals. Scholars look for datasets to compare their own results with legacy data, to prevent repeating cost-intensive experiments or to merge multiple data sources into a new dataset, in order to explore novel hypotheses. Biodiversity research is one example of a data-intensive research area. It examines the variety of species and their genetic and ecological diversity [Lo10]. Multiple studies report on various obstacles users encounter while searching for datasets. Main problems are missing primary data and lacking information on data collection methods [Pa16], scattered data in different data repositories [Cu18], unsatisfactory user interfaces [Gr20] and insufficient metadata descriptions and unaligned terminologies [Lö21]. Semantic search approaches that are going beyond the classical keyword search can partially help

---

<sup>1</sup> Heinz Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University, Jena, Germany, felicitas.loeffler@uni-jena.de

<sup>2</sup> Heinz Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University, Jena, Germany

<sup>3</sup> Semantic Software Lab, Concordia University, Montréal, Canada

<sup>4</sup> Heinz Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University, Jena, Germany  
Michael-Stifel-Center for Data-Driven and Simulation Science, Jena, Germany  
German Center for Integrative Biodiversity Research (iDiv), Halle-Jena-Leipzig, Germany

<sup>5</sup> Data Acquisition and Mobilization Department, Institute of Data Science, German Aerospace Center (DLR), Jena, Germany

to overcome the current drawbacks: In particular, using standardized domain vocabularies and terminologies in the search process enhances sparse metadata and enables users to find more relevant data [LK16]. However, very few semantic search solutions for dataset search are publicly available. In this work, we present a semantic search over metadata files and domain vocabularies. We link entries in metadata files with concepts from relevant domain taxonomies and ontologies with respective URIs. Based on the linked vocabularies and its concepts, we also derive an entity type (entity category or type). Both information (URIs and entity types) are added to metadata as semantic annotations and are utilized in a semantic index with a URI-based retrieval model. In order to study which search input users prefer in dataset search, we implemented two user interfaces: a UI with a single input field and a second UI enabling a search over different semantic categories. In both interfaces, the search results are enhanced with text highlightings and additional information on demand, to support users in obtaining a quick overview with explanations of the returned datasets. We evaluated the system with 20 scholars working in the field of Life Sciences and Environmental Sciences. The participants performed various search tasks in a given time frame and answered questionnaires before and after each search task, for each user interface, and at the end of the session. Thus, our contribution is two-fold:

1. We present an architecture enabling a flexible, semantic dataset search; and
2. We provide a usability study on two main aspects in search: the query input and search result explanations, such as textual highlightings with additional information.

The structure of the paper is as follows: First, we present related work in Section 2, followed by the evaluation setup in Section 3, including information on our system architecture. The evaluation results are presented in Section 4.

## 2 Related Work

Due to the multitude of heterogenous data formats, research data, such as spread sheets, multimedia files or questionnaires, are described by metadata. This descriptive information on, e.g., author, title, abstract, collection time and data format, are utilized by data portals in search indexes [KCW18]. Very few usability studies for dataset search are publicly available. Therefore, besides a discussion on user studies in dataset retrieval in the Life Sciences, here we introduce semantic search approaches in the Life Sciences and user studies for semantic search systems.

**User studies in dataset retrieval.** [Di17] and [Ch18] are examples for user studies in the Datamed portal,<sup>6</sup> a federated approach providing datasets from numerous biomedical data repositories. A second data portal that conducted a user evaluation is DataONE<sup>7</sup> [Vo15],

---

<sup>6</sup> Datamed, <https://datamed.org/>

<sup>7</sup> DataONE, <https://www.dataone.org/>

a data portal providing environmental and biological datasets from various sources. The studies by [MM15] and [Ka20, Ka21] report results from user studies in the earth observation and oceanography domain (dealing with spatial and temporal data). Most of the studies collected qualitative data with less than 30 subjects. All studies focused on the evaluation of the usability and utilized various usability methods, such as thinking-aloud, questionnaires and interviews. Apart from [Ch18], all studies observe two main obstacles: (A) metadata quality [Vo15, Di17, Ka20, MM15], e.g., different spellings of variable names and (B) problems with the user interface, e.g., inconsistent information in title and description or a purely text-based presentation [Vo15, Ka20]. Users of the Datamed portal also suggested offering an enhanced search input, allowing them to search for domain specific topics, such as a search for phenotypes or genes [Di17]. Another source for guidelines and good practices in dataset search are the outcomes of the RDA Data Discovery Paradigms Interest Group.<sup>8</sup> Based on 79 data discovery use cases, heuristic evaluations and interviews with experts, they propose ten recommendations for enhancing dataset search and user experience [Wu19]. The main areas addressed in the RDA's suggestions are: (i) providing multiple search inputs in order to support different information needs, (ii) filtering options, and (iii) comprehensible search summaries on the search entry result page (SERP), by displaying dataset snippets that belong to a search query.

**Semantic search systems in the Life Sciences.** Semantic search is a “search beyond keywords” [BBH16]. Instead of matching query input and document content syntactically, the result set also contains semantically related content by exploiting additional knowledge to the search process. Data sources are either plain text, structured data from knowledge bases or a combination of both. Search approaches vary between *keyword search* (+ query expansion techniques), *structured search* by using specific query languages such as SPARQL and *full natural language questions* in question answering [BBH16]. Multiple semantic search approaches have emerged in the biomedical domain focusing on scientific articles from PubMed<sup>9</sup>, e.g., [LLW15, Mu17, A118, SPA18]. For dataset search, only very few semantic systems exist such as BioFid [Pa21] (German legacy collection data), Datamed [Ch18] (biomedical datasets) and GFBio (biological and environmental data) [Lö17]. Most systems expand the query input. Data discovery with named entities over semantic categories is only supported by [LLW15, SPA18, A118]. The search summaries usually present snippets of the document or datasets, contain information on data sources and highlight the matched search terms. Explanations on semantic categories or matching entities [SPA18, A118], full query information [Ch18] or additional information from external sources such as wikipedia [Pa21] are only partially available.

**User studies on semantic search systems.** [KB07] was one of the first studies exploring different query interfaces on a knowledge base with geographical information. This study also utilized a combination of search tasks and a System Usability Scale (SUS) questionnaire [Br96] to determine which kind of query input is useful for a search over knowledge

---

<sup>8</sup> RDA DDP IG, <https://rd-alliance.org/node/52248/outputs>

<sup>9</sup> PubMed, <https://pubmed.ncbi.nlm.nih.gov>

bases. The user interface with support for full natural language questions obtained the best success rate and resulted in the highest SUS score (75.73). The authors of [EWC12] utilized the same geographic knowledge base and analyzed two user types, experts and casual, for three different query inputs: natural language, form-based and graph-based. Their results show that casual users prefer the form-based query interface and experts favored the graph-based search. The study by [Ve16] also conducted an A/B test with 15 users and two semantic search systems over an RDF dataset with administrative and financial information of Norwegian companies. The participants performed four search tasks for each system and filled in questionnaires before and after each search task, after each system, and at the very end of the evaluation. Similar to the study of [EWC12], the form-based system was favored by casual users and the graph-based one by expert users.

The evaluation studies in dataset retrieval introduced above only collected qualitative data, through questionnaires and interviews. In contrast, the semantic search user studies followed the Text REtrieval Conference (TREC) guidelines for interactive information retrieval [Du05], but did not focus on dataset search. [Ch18] and [Lö17] are dataset search approaches with semantic enhancements based on query expansion. While [Ch18] extends the entered keywords with synonyms, scientific and common names obtained from the UMLS,<sup>10</sup> a knowledge base for the biomedical domain, [Lö17] utilizes the GFBio TS [Ka14]. Both systems do not offer searching for specific entity types and only partially support explanations on matched entities.

### 3 Evaluation Setup

Previous studies always examined user interfaces as a whole. In order to identify the impact of different UI choices more precisely, we decided to focus our study on two main parts of search: the query input and the search result summary. These two aspects are also top-ranked recommendations in the RDA guidelines [Wu19]. In the user study of [Di17], users mentioned the need for searching specific entity types. Our previous work [Lö21] also revealed that scientific information needs to differ in granularity. Search questions can be very specific, e.g., a search for a concrete species, or can have a broader scope, e.g., a search for datasets with organisms in water samples. Therefore, we implemented two user interfaces: the first one (Biodiv 1) provides a category-based input of search terms and the second interface (Biodiv 2) offers a classical single-input field and determines entity types automatically. The search results in both interfaces can contain datasets going beyond the entered query terms, to broaden the search on semantically related terms, like synonyms. Concerning the presentation of the search results, we followed the RDA recommendations for search summaries in dataset search. To ensure a consistent layout, we aligned the presentation of each dataset entry to existing data portals, such as GFBio<sup>11</sup> and Zenodo.<sup>12</sup> In a previous study [LK16], we determined that end users need more explanations

<sup>10</sup> UMLS, <https://www.nlm.nih.gov/research/umls/index.html>

<sup>11</sup> GFBio, <https://www.gfbio.org>

<sup>12</sup> Zenodo, <https://zenodo.org/>

when being faced with search results going beyond keywords. This motivated us to further focus our analysis of the search summary to the presentation of explanations, allowing us to study whether displaying terminologies and matching URIs confuse and distract users or help them. Through the biodiversity research projects we are involved in (e.g., iDiv,<sup>13</sup> NFDI4Biodiversity<sup>14</sup>), we know that most scholars in the Life and Environmental Sciences are casual users, with no expertise in semantic technologies. However, as FAIR data management with terminologies are increasingly forming the semantic backbone in academia, it is necessary to explore what additional information end users need from a semantic dataset search, in order to understand and assess the relevance of search results.

### 3.1 System Architecture

The overall architecture (Figure 1) is an extended version of a framework we introduced in previous work [Sh21]. According to [BBH16], our system represents a structured search over text and knowledge bases. A semantic search index (see paragraph below on semantic indexing) and a local terminology service with ontologies of the OBO Foundry initiative<sup>15</sup> form the back-end. In order to link entered keywords with concepts in ontologies, we utilize an updated version of the Semantic Assistants framework [WG08]. This new version provides several text mining pipelines from various sources in a Spring boot application and two micro services, accessible via REST services.<sup>16</sup> The first micro service offers pipelines of the text mining framework GATE.<sup>17</sup> Currently, three taggers are available: GATE ANNIE [Cu13], which extracts named entities, such as people, locations and organizations; the BiodivTagger [Lö20], focusing on the recognition of environmental terms, data parameters, materials, chemicals and processes; and the OrganismTagger [Na11] extracting species. A REST API in the middleware enables the front-end to communicate with the back-end applications.<sup>18</sup>

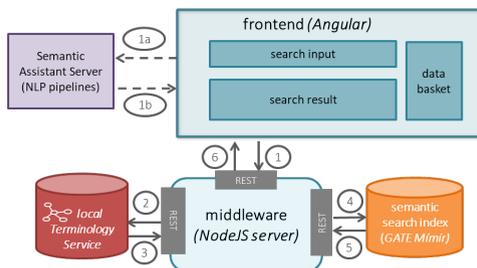


Fig. 1: Extended architecture and overall flow, based on the previous version introduced in [Sh21]

<sup>13</sup> iDiv, <https://www.idiv.de>

<sup>14</sup> NFDI4Biodiversity, <https://www.nfdi4biodiversity.org/de/>

<sup>15</sup> OBO Foundry, <https://obofoundry.org/>

<sup>16</sup> SA2.0, <https://github.com/fusion-jena/SA2.0>

<sup>17</sup> GATE, <https://gate.ac.uk/>

<sup>18</sup> [Dai:Si] semantic search, <https://github.com/fusion-jena/daisi-semantic-search>

Users can either enter keywords per category or they can type all search terms in one input field (1). In the latter case, the Semantic Assistants service is called to look for matching URIs and entity types in the entered search terms (1A–1B). Afterwards, the query terms are sent to the terminology service (2) and matching URIs are returned to the middleware (3). The obtained URIs and entity types are utilized to fill templates being sent to the search engine (4). The obtained datasets from the search index (5) are forwarded to the Angular application, and finally, the result is presented to the user (6).

**Dataset corpus preparation and semantic indexing.** We downloaded metadata files from GFBio<sup>19</sup> being relevant for the selected search tasks. The files were provided in a repository specific metadata format.<sup>20</sup> We manually annotated the search tasks with URIs from OBO Foundry ontologies to obtain descendants nodes, labels and alternate labels via SPARQL queries from our local terminology service. These additional terms were added to the original query terms. We downloaded only the top-100 datasets per query, to ensure a manageable corpus size. The obtained ~52.000 metadata files were semantically annotated with the OrganismTagger [Na11] and the BiodivTagger [Lö20]. As a result, key terms in the metadata were linked to matching entities and their types in domain specific ontologies, such as Organism, Environment, Data Parameter, Process, or Material. Using GATE’s Semantic Enrichment Processing Resource, we added ancestor nodes to each entity as additional annotations (subClassOf\* relations), facilitating a hierarchy search at run-time. SPARQL queries for hierarchy relations can become complex, and a call to a terminology service can take up to several minutes to come back. Therefore, we added these hierarchical relations to the metadata files in the pre-processing phase as ‘broader’ annotations. All metadata files were indexed with GATE Mimir [Cu13], a search engine that provides indexing of semantic annotations with entity types and features, like matching URIs.

**User interfaces.** We proposed several UIs in clickable paper prototypes, discussing them in a focus group with three biodiversity scholars. We finally selected two UIs for supporting scholars who are not experts in semantic web technologies. As a result, we decided to support keyword search in both user interfaces and to omit an entity-based search. The first UI (Biodiv 1, Figure 2 (left)) provides a category-based input. Here, users need to actively sort the search terms into given domain categories. The entered keywords are sent to the terminology service to find matching URIs. In the second UI (Biodiv 2, Figure 2 (right)), the query terms are additionally sent to the Semantic Assistants server to obtain the entity types. The retrieved URIs and entity types are then utilized to form the final query being sent to the search index. In case no matching URI can be found, we also consider the original keywords in the query (Listing 1).

```
((('honeybee' IN {Organism}) OR ('honeybee' OVER {Organism}) OR ('honeybee' AND {Organism})) OR ({Organism broader='`http://purl.obolibrary.org/obo/NCBITaxon_7460'}` OR {Organism inst='`http://purl.obolibrary.org/obo/NCBITaxon_7460'}`}))
```

List. 1: Full query for the search ‘honeybee’ in Figure 2

<sup>19</sup> GFBio Metadata, <https://github.com/fusion-jena/GFBioMetadata>

<sup>20</sup> PAN-MD, <http://ws.pangaea.de/schemas/pangaea/MetaData.xsd>

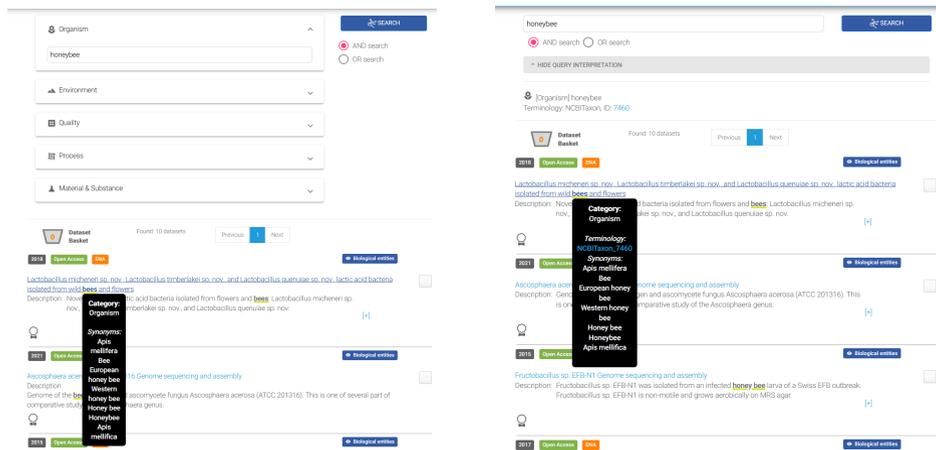


Fig. 2: Screenshots of Biodiv 1 and Biodiv 2, with a search for ‘honeybee’, returning alternate labels

We leverage search templates with the introduced ‘broader’ annotation to consider exact matches, as well as more specific terms, in the result set. The search results display also provides two types of highlighting: The original entered query terms and their related terms are highlighted in bold font (‘default highlightings’). If synonyms are available, they are displayed with a green underline. On demand (mouse-over) a separate dialog opens and displays alternate labels. In Biodiv 1, this dialog shows only the entity type of the highlighted term, whereas in Biodiv 2, the respective terminologies and URIs are presented additionally. A summarized version of the available synonyms is listed in an explanation tab. This tab also displays information on the search query. In Biodiv 1, users can view only a shortened search query, whereas in Biodiv 2, the full query to the search index is presented. Supplementary highlightings of further biological entities can be displayed with a ‘biological entities’ button. This function sends the textual information of a dataset to the Semantic Assistants service and returns annotations obtained from the taggers. These highlightings of data parameters, environmental terms, processes, materials and species are underlined in blue color.

### 3.2 Experimental Design

Our setup generally follows the TREC-9 guidelines.<sup>21</sup> The overall aim was to measure the usability of a semantic dataset search, but with a particular focus on the query input and provided explanations in the search summary. In total, 20 scholars with a research background in the Life Sciences and Environmental Sciences took part. Each participant performed eight search tasks in different orderings, e.g., “*What data exists in the repository*

<sup>21</sup> TREC Interactive Track, <https://trec.nist.gov/data/t9i/spec.html>

for bacteria in the groundwater?”. For each user interface, every scholar carried out four tasks. In a maximum of five minutes, the users had to search for up to three datasets per task. When they considered a dataset as relevant, they added the dataset to the data basket, and at the end of the five minutes, the basket was downloaded. We conducted the evaluation in live sessions at the Friedrich Schiller University Jena (FSU Jena) or visited the participants at their working places, e.g., iDiv, Leipzig<sup>22</sup>, Senckenberg, Frankfurt<sup>23</sup> and BGBM, Berlin.<sup>24</sup> The total evaluation time for each session was around 120 minutes. The search part took around 80 minutes, with ten minutes per task. The non-search part was around 40 minutes; 25 minutes were utilized for the post system questionnaires and five minutes for the exit questionnaire, with questions about a comparison between the two interfaces and some demographic questions. Before the start, we allotted 15 minutes for introductory explanations. For instance, we explained the purpose of the study and demonstrated some example searches to minimize the training effect [RC08].

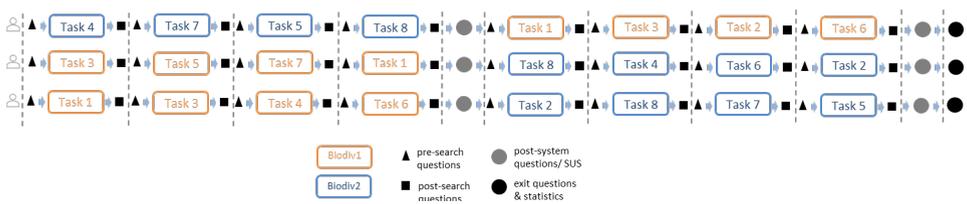


Fig. 3: Study flow for the first three users. Every user performed eight search tasks in different orderings. Half of the participants started with Biodiv 1, the other half started with Biodiv2.

**Data collection and metric.** Measuring the usability involves examining whether users are able to complete tasks in a given time (efficiency), how easy it is to work with the system (learnability), how many errors occur (issues), how easy it is to remember navigations and functions (memorability) and how satisfied users are overall with a system (satisfaction) [Ni93]. Apart from the memorability, we addressed all variables in our study. We evaluated the efficiency and effectiveness of the query input by means of user tasks (performance-based metrics). If users found three datasets in five minutes, we counted it as a complete task; if only one or two datasets could be found, it was considered as partially complete; and if no dataset was found, it counted as an uncompleted task. We measured the satisfaction, the learnability and collected usability issues with questionnaires (self-reported metrics) and through observation. Concerning the highlightings and explanations, we explored the comprehensibility, completeness and satisfaction through questionnaires. We also observed the participants and encouraged them to report occurring issues orally.

**Search tasks and questionnaires.** We selected eight search tasks from the question corpus we introduced in our previous work, to ensure that relevant categories from biodiversity research are contained [Lö21]. In order to better guide users through the evaluation, we

<sup>22</sup> iDiv, <https://www.idiv.de/>

<sup>23</sup> Senckenberg, <https://www.senckenberg.de>

<sup>24</sup> BGBM, <https://www.bgbm.org>

prepared a survey for each user.<sup>25</sup> The overall evaluation flow is presented in Figure 3. The survey provided questions before and after each task, according to the TREC guidelines, e.g., whether the participants expect certain content in the datasets, as well as questions on satisfaction, easy of use and ease of learnability afterwards. After four tasks (=each user interface), the scholars had to assess ten statements (a SUS questionnaire) on a five-point Likert scale to capture feedback on the usability. However, as we aimed to study the query input and explanation strategies, we adapted the classical SUS statements to our needs. The final SUS questionnaire contained two questions on the query input, three questions on the default highlightings, three questions concerning the biological entities highlighting and two questions on the provided query explanations. The final search tasks, the questionnaires and the original survey result files are available in Zenodo [Lö22].

## 4 Results

We compiled all results into one large CSV file and created a Jupyter notebook<sup>26</sup> to analyze the results. The code and the full results are available on GitHub.<sup>27</sup> Two-third of the 20 scholars search for datasets monthly or at least once in a year, and the other seven scholars use dataset search applications daily or weekly. The overall SUS scores for both interfaces resulted in values above 68 (Biodiv 1: 68, Biodiv 2: 71), which points to a good usability with respect to the two studied UI components in both interfaces. However, the dispersion in Biodiv 2 is large, because eleven users gave higher ratings for the interface with the single input field (Biodiv 2) than for the category-based search (Biodiv 1).

**Search Input.** With respect to the task success and task time, more scholars were able to retrieve three datasets, with fewer failure cases and in a shorter time, with Biodiv 2 (204ms) than with Biodiv 1 (225ms). However, the differences are small. Figure 4 reveals that for almost all tasks, the users were able to retrieve datasets. For two tasks, the success rate is low in both interfaces and the users complained that the returned datasets were not relevant or only partial information was relevant. This points to missing data in the metadata corpus. In addition, most scholars were not familiar with the search tasks (see the figures available on GitHub<sup>28</sup>). As the provided search tasks addressed a specific research question or a scientific information need, we did not expect that. However, this lack of specific knowledge impacted the results.

Concerning the learnability (see the figures on GitHub), the results of the questionnaires after each task and the respective statements in the SUS questionnaire reveal that it was easier for users to get started with Biodiv 2. But with respect to the ease of use, there is no

---

<sup>25</sup> Limesurvey, <https://www.limesurvey.org>

<sup>26</sup> Jupyter notebook, <https://jupyter.org/>

<sup>27</sup> Analysis, <https://github.com/fusion-jena/semantic-search-usability-analysis>

<sup>28</sup> Results for 20 users, <https://github.com/fusion-jena/semantic-search-usability-analysis/tree/main/analysis/results20>

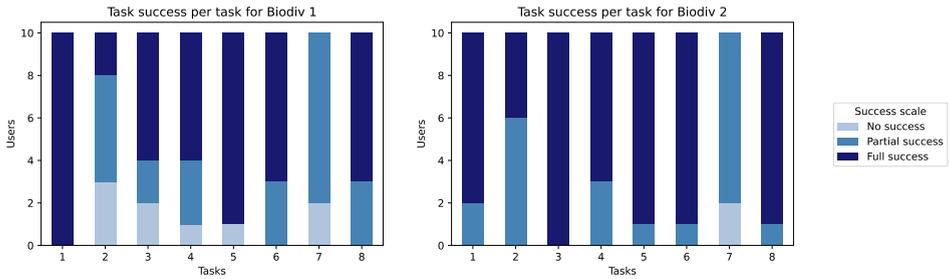


Fig. 4: Task success per task

difference between the two interfaces. Looking at the answers in the exit questionnaire paints the same picture. The pre-defined categories in Biodiv 1 were not as easy to understand, and it took some time for users to become familiar with the topical categorization of search terms. However, comments on Biodiv 1 reveal that one half of the users liked the category-based interface, because categories “helped to narrow down the search criteria and to get pertinent results.” In contrast, the other half of scholars liked Biodiv 2, because it is “easier, as it goes straight forward without thinking about categories,” and it “is more general and helped in searching for topics that I was unfamiliar with.”

**Highlightings and explanations.** The users gave high ratings for the overall usefulness of default highlightings in both interfaces (Figure 5). However, in some result sets, the default highlightings were missing or too many terms were highlighted, which led to medium ratings. Overall, the provided information is sufficient and comprehensible. Concerning the highlighting of biological entities in the datasets (see figure on GitHub), we observed that only 50% of the participants utilized this function. Only when they had to give a rating on this function in the SUS questionnaire, they investigated it. Thus, this might have led to medium ratings with respect to the comprehensibility. However, the participants gave high ratings for the overall usefulness of this additional highlighting. For Biodiv 2, the users pointed out more than twice as often as for Biodiv 1 that they would need more information on the presented information of the biological entities. This also correlates with our observations. In some cases, the biological entity function took some time to deliver a result (as it was a request to the Semantic Assistant service calling various NLP pipelines) or not all expected terms were highlighted. In these cases, users looked up the terms in Google<sup>29</sup> or Wikipedia<sup>30</sup> (which was permitted), to obtain more information. With respect to the usage of the query explanation tab, we noticed less usage, too. Thus, the helpfulness obtained medium ratings (see figure on GitHub). Concerning the comprehensibility, users gave a little more preference to the simpler query explanations without URI information in Biodiv 1.

<sup>29</sup> Google, <https://www.google.de/>

<sup>30</sup> Wikipedia, <https://de.wikipedia.org>

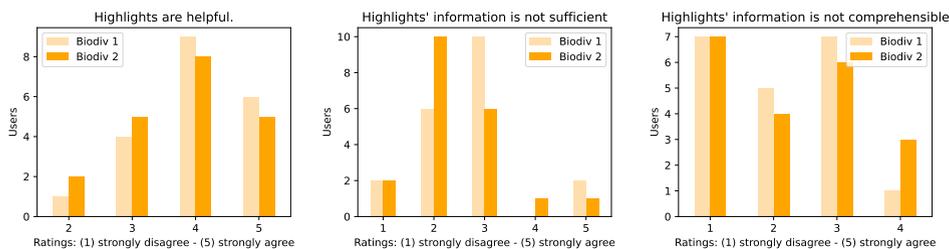


Fig. 5: SUS statements on the default highlighting of query terms in bold font

In comparison to other user studies [EWC12, Ve16], our results show that overall both interfaces are suitable for a semantic dataset search in biodiversity research. A user interface with a single input field and automatic category detection is more suitable when users have no expertise in the search topic. In order to get more insights on daily usage of the systems with the user's own search tasks, further long-term studies are needed. The users appreciated highlightings and explanations and were not confused about information on terminologies and URIs. Only with respect to query information, users preferred the simple explanations without information on the linked ontology. Concerning further improvements of the user interface, the participants suggested to integrate facets and filters to narrow down the results, e.g., to sort the data along metadata fields such as data repository or data type. This confirms the RDA recommendations [Wu19] that users need different entry points and filtering options in dataset search.

## 5 Conclusion

In this study, we proposed an improved user interface for a semantic search over datasets in the Life Sciences. We conducted a usability study of this novel system with two user interfaces, with a particular focus on query inputs and different explanation strategies. Our results reveal that both interfaces are suitable for semantic dataset search. For users that are not familiar with a search topic, a user interface with a single input field is slightly more efficient. Details about utilized ontologies and URIs are helpful and desired. More studies in other applied domains are needed to examine whether this outcome can be generalized. In addition, long-term studies would be beneficial to study semantic search in daily usage.

## Acknowledgement

We would like to thank all participants in this study for their time and valuable feedback.

## Bibliography

- [Al18] Allot, Alexis; Peng, Yifan; Wei, Chih-Hsuan; Lee, Kyubum; Phan, Lon; Lu, Zhiyong: LitVar: a semantic search engine for linking genomic variant data in PubMed and PMC. *Nucleic Acids Research*, 46(W1):W530–W536, 2018.
- [BBH16] Bast, Hannah; Buchhold, Björn; Haussmann, Elmar: Semantic Search on Text and Knowledge Bases. *Foundations and Trends® in Information Retrieval*, 10(2-3):119–271, 2016.
- [Br96] Brooke, John: SUS - A quick and dirty usability scale. In (Jordan, P.W.; Thomas, B.; McClelland, I.L.; Weerdmeester, B., eds): *Usability Evaluation In Industry*. CRC Press, chapter SUS - A quick and dirty usability scale, 1996.
- [Ch18] Chen, Xiaoling; Gururaj, Anupama E; Ozyurt, Burak; Liu, Ruiling; Soysal, Ergin; Cohen, Trevor; Tiryaki, Firat; Li, Yueling; Zong, Nansu; Jiang, Min; Rogith, Deevakar; Salimi, Mandana; Kim, Hyeon-eui; Rocca-Serra, Philippe; Gonzalez-Beltran, Alejandra; Farcas, Claudiu; Johnson, Todd; Margolis, Ron; Alter, George; Sansone, Susanna-Assunta; Fore, Ian M; Ohno-Machado, Lucila; Grethe, Jeffrey S; Xu, Hua: DataMed – an open source discovery index for finding biomedical datasets. *Journal of the American Medical Informatics Association*, 25(3):300–308, 2018.
- [Cu13] Cunningham, Hamish; Tablan, Valentin; Roberts, Angus; Bontcheva, Kalina: Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics. *PLoS Computational Biology*, 9(2):e1002854–e1002854, 2013.
- [Cu18] Culina, Antica; Baglioni, Miriam; Crowther, Tom W.; Visser, Marcel E.; Woutersen-Windhouver, Saskia; Manghi, Paolo: Navigating the unfolding open data landscape in ecology and evolution. *Nature Ecology & Evolution*, 2(3):420–426, 2018.
- [Di17] Dixit, Ram; Rogith, Deevakar; Narayana, Vidya; Salimi, Mandana; Gururaj, Anupama; Ohno-Machado, Lucila; Xu, Hua; Johnson, Todd R: User needs analysis and usability assessment of DataMed – a biomedical data discovery index. *Journal of the American Medical Informatics Association*, 25(3):337–344, 2017.
- [Du05] Dumais, Susan: *The Interactive TREC Track: Putting the User Into Search*. MIT Press, 2005.
- [EWC12] Elbedweihy, Khadija; Wrigley, Stuart N.; Ciravegna, Fabio: Evaluating Semantic Search Query Approaches with Expert and Casual Users. In (Cudré-Mauroux, Philippe; Heflin, Jeff; Sirin, Evren; Tudorache, Tania; Euzenat, Jérôme; Hauswirth, Manfred; Parreira, Josiane Xavier; Hendler, Jim; Schreiber, Guus; Bernstein, Abraham; Blomqvist, Eva, eds): *The Semantic Web – ISWC 2012*. Springer Berlin Heidelberg, pp. 274–286, 2012.
- [Gr20] Gregory, Kathleen; Groth, Paul; Scharnhorst, Andrea; Wyatt, Sally: Lost or Found? Discovering Data Needed for Research. *Harvard Data Science Review*, 4 2020. <https://doi.org/10.1162/99608f92.e38165eb>.
- [Ka14] Karam, Naouel.; Fichtmüller, David.; Gleisberg, Maren.; Becker, Florian.; Tolksdorf, Robert.; Müller-Birn, Claudia.; Paschke, Adrian.; Güntsch, Anton. (Eds.): , *The Terminology Service of the German Federation for Biological Data (GFBio) - Service of semantic technologies in scientific environments*. <http://terminologies.gfbio.org/> - [Accessed 2023-01-22], 2014.

- [Ka20] Kalantari, Mohsen; Syahrudin, Syahrudin; Rajabifard, Abbas; Subagyo, Hardi; Hubbard, Hannah: Spatial Metadata Usability Evaluation. *ISPRS International Journal of Geo-Information*, 9(7), 2020.
- [Ka21] Kalantari, Mohsen; Syahrudin, Syahrudin; Rajabifard, Abbas; Hubbard, Hannah: Synchronising Spatial Metadata Records and Interfaces to Improve the Usability of Metadata Systems. *ISPRS International Journal of Geo-Information*, 10(6), 2021.
- [KB07] Kaufmann, Esther; Bernstein, Abraham: How Useful Are Natural Language Interfaces to the Semantic Web for Casual End-Users? In (Aberer, Karl; Choi, Key-Sun; Noy, Natasha; Allemang, Dean; Lee, Kyung-Il; Nixon, Lyndon; Golbeck, Jennifer; Mika, Peter; Maynard, Diana; Mizoguchi, Riichiro; Schreiber, Guus; Cudré-Mauroux, Philippe, eds): *The Semantic Web*. Springer Berlin Heidelberg, pp. 281–294, 2007.
- [KCW18] Khalsa, SiriJodha; Cotroneo, Peter; Wu, Mingfang: A survey of current practices in data search services. Technical report, Research Data Alliance Data (RDA) Discovery Paradigms Interest Group, 2018.
- [LK16] Löffler, Felicitas; Klan, Friederike: Does Term Expansion Matter for the Retrieval of Biodiversity Data? In (Martin, Michael; Cuquet, Martí; Folmer, Erwin, eds): *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16)*, co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016). CEUR Workshop Proceedings, 2016.
- [LLW15] Liu, Yifeng; Liang, Yongjie; Wishart, David: PolySearch2: a significantly improved text-mining system for discovering associations between human diseases, genes, drugs, metabolites, toxins and more. *Nucleic acids research*, 43:W535–42, 2015.
- [Lo10] Loreau, Michel: *Excellence in Ecology: Book 17, The Challenges of Biodiversity Science*. International Ecology Institute, Oldendorf, Germany, 2010.
- [Lö17] Löffler, Felicitas; Opasjumruskit, Kobkaew; Karam, Naouel; Fichtmüller, David; Schindler, Uwe; Klan, Friederike; Müller-Birn, Claudia; Diepenbroek, Michael: Honey Bee Versus *Apis Mellifera*: A Semantic Search for Biological Data. In (Blomqvist, Eva; Hose, Katja; Paulheim, Heiko; Ławrynowicz, Agnieszka; Ciravegna, Fabio; Hartig, Olaf, eds): *The Semantic Web: ESWC 2017 Satellite Events: Portorož, Slovenia*. Springer International Publishing, pp. 98–103, 2017.
- [Lö20] Löffler, Felicitas; Abdelmageed, Nora; Babalou, Samira; Kaur, Pawandee; König-Ries, Birgitta: Tag Me If You Can! Semantic Annotation of Biodiversity Metadata with the QEMP Corpus and the BiodivTagger. In: *Proceedings of The 12th Language Resources and Evaluation Conference*. European Language Resources Association, pp. 4557–4564, 2020.
- [Lö21] Löffler, Felicitas; Wesp, Valentin; König-Ries, Birgitta; Klan, Friederike: Dataset search in biodiversity research: Do metadata in data repositories reflect scholarly information needs? *PLOS ONE*, 16(3):1–36, 2021.
- [Lö22] Löffler, Felicitas; Shafiei, Fateme; Witte, René; König-Ries, Birgitta; Klan, Friederike: [Dataset] Supplementary material for a usability evaluation of a semantic search for biological datasets, <https://doi.org/10.5281/zenodo.7388037>, 2022.

- [MM15] Megler, Veronica M.; Maier, David: Are Data Sets Like Documents?: Evaluating Similarity-Based Ranked Search over Scientific Data. *TKDE: Transactions on Knowledge and Data Engineering*, 27(1), 2015.
- [Mu17] Mueller, Bernd; Poley, Christoph; Pössel, Jana; Hagelstein, Alexandra; Gübitz, Thomas: LIVIVO - the Vertical Search Engine for Life Sciences. *Datenbank-Spektrum*, 17(1):29–34, 2017.
- [Na11] Naderi, Nona; Kappler, Thomas; Baker, Christopher J. O.; Witte, René: OrganismTagger: detection, normalization and grounding of organism entities in biomedical documents. *Bioinformatics*, 27(19):2721–2729, 2011.
- [Ni93] Nielsen, Jakob: Chapter 6 - Usability Testing. In (NIELSEN, JAKOB, ed.): *Usability Engineering*, pp. 165–206. Morgan Kaufmann, 1993.
- [Pa16] Parker, Timothy H.; Forstmeier, Wolfgang; Koricheva, Julia; Fidler, Fiona; Hadfield, Jarrod D.; Chee, Yung En; Kelly, Clint D.; Gurevitch, Jessica; Nakagawa, Shinichi: Transparency in Ecology and Evolution: Real Problems, Real Solutions. *Trends in Ecology & Evolution*, 31(9):711 – 719, 2016.
- [Pa21] Pachzelt, Adrian; Kasperek, Gerwin; Lücking, Andy; Abrami, Giuseppe; Driller, Christine: Semantic Search in Legacy Biodiversity Literature: Integrating data from different data infrastructures. *Biodiversity Information Science and Standards*, 5:e74251, 2021.
- [RC08] Rubin, Jeffrey; Chisnell, Dana: *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, 2nd Edition. Wiley, 2008.
- [Sh21] Shafiei, Fateme; Löffler, Felicitas; Thiel, Sven; Opasjumruskit, Kobkaew; Grabiger, Denis; Rauh, Pauline; König-Ries, Birgitta: [Dai:Si] - A Modular Dataset Retrieval Framework with a Semantic Search for Biological Data. In (Sanfilippo, Emilio M.; Kutz, Oliver; Troquard, Nicolas; Hahmann, Torsten; Masolo, Claudio; Hoehndorf, Robert; Vita, Randi; Algergawy, Alsayed; Karam, Naouel; Klan, Friederike; Michel, Franck; Rosati, Ilaria, eds): *S4BioDiv 2021: 3rd International Workshop on Semantics for Biodiversity*, held at JOWO 2021: Episode VII The Bolzano Summer of Knowledge, September 11–18, 2021, Bolzano, Italy. 2021.
- [SPA18] Soto, Axel J; Przybyła, Piotr; Ananiadou, Sophia: Thalia: semantic search engine for biomedical abstracts. *Bioinformatics*, 35(10):1799–1801, 2018.
- [Ve16] Vega-Gorgojo, Guillermo; Slaughter, Laura; Giese, Martin; Heggstøyl, Simen; Soylu, Ahmet; Waaler, Arild: Visual query interfaces for semantic datasets: An evaluation study. *Journal of Web Semantics*, 39:81–96, 2016.
- [Vo15] Volentine, Rachel; Owens, Amber; Tenopir, Carol; Frame, Mike: Usability Testing to Improve Research Data Services. *Qualitative and Quantitative Methods in Libraries*, 4(1):59–68, 2015.
- [WG08] Witte, René; Gitzinger, Thomas: Semantic Assistants – User-Centric Natural Language Processing Services for Desktop Clients. In (Domingue, John; Anutariya, Chutiporn, eds): *The Semantic Web*. Springer Berlin Heidelberg, pp. 360–374, 2008.
- [Wu19] Wu, Mingfang; Psomopoulos, Fotis; Khalsa, Siri Jodha; de Waard, Anita: Data Discovery Paradigms: User Requirements and Recommendations for Data Repositories. *Data Science Journal*, 18(1):3, 2019.

# ReStoRunT: Simple Recording, Storing, Running and Tracing changes in Spreadsheets<sup>1</sup>

Wolfgang Müller<sup>2</sup>; Lukrécia Mertová<sup>2</sup>

**Abstract:** In addition to the ubiquitous *big data*, one key challenge in data processing and management in the life sciences is the *diversity of small data*. Diverse pieces of small data have to be transformed into standards-compliant data. Here, the challenge lies not in the difficulty of single steps that need to be performed, but rather in the fact that many transformation tasks are to be performed once or only a few times. This limits the time that can be put into automated approaches, which in turn severely limits the verifiability of such transformations. As much of the data to be processed is stored in spreadsheets, within this paper we justify and propose a lightweight recording-based solution that works on a wide variety of spreadsheet programs, from Microsoft Excel to Google Docs.

**Keywords:** Provenance; Harmonisation; Spreadsheets

## 1 Introduction

One of the challenges of real-life data harmonisation in the life sciences is the implementation of standards in everyday work. The challenge lies in the fact that research needs to be flexible and fast, while in the end, one needs reliable data with known semantics. This is the gist of the FAIR principles [Wi16] - Findability, Accessibility, Interoperability and Reusability, which depend mostly on the known semantics of the data.

The semantics of the data is typically conveyed in one of three ways

1. Annotation to ontologies (for example, using web standards like the Resource Description Framework [CK04])
2. Description via markup languages (using SBML [Hu03], for example)
3. Via location in a spreadsheet (as done by many MIBBI [FA22] standards that provide mandatory sets of attributes and sometimes even precise file formats to be filled)

In the latter two cases, the semantics is not conveyed via an ontology but rather via the documentation of the respective formats.

---

<sup>1</sup> Supported by the Heidelberg Institute for Theoretical Studies and the Klaus Tschira Foundation, as well as MESI-STRAT. MESI-STRAT has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 754688.

<sup>2</sup> Heidelberg Institute for Theoretical Studies — HITS gGmbH, 69118 Heidelberg, Germany wolfgang.mueller@h-its.org

CSV files and spreadsheets play an essential role here. They are used as lightweight databases with bespoke data models and are transformed towards standard formats to convey semantics by adhering to a pre-defined structure.

As a matter of efficiency, scientists typically use formats compatible with the machines and the software they are using. They are usually different to standard formats such as MIBBI formats. Scientists design their everyday formats to be simple to use with their machines and their software. Often the formats are organized around one cut-and-paste operation from a key proprietary software. This way of action reduces errors and minimizes time for an outcome.

Standards like MIBBI provide mandatory sets of attributes or concrete file formats for data files. However, scientists are left alone with the question of how to bring their day-to-day files into a common format (*e.g.* for one partner or the local lab) or a mandatory format (*e.g.* a standard format for a data publication).

Both transformation tasks are indeed very similar. They mainly differ in the number of files concerned. Long-term experience with tools like RightField [Wo11] or openRefine [te22] showed that a given internal file format is typically used only a couple of times, a common format is used a couple of tens of times, and finally, the mandatory format is used thousands of times.

The difficulty in this setup does not lie in the transformations themselves. They are mostly based on simple operations on single values (such as moving a value to another cell) or tables (such as the transposition of a matrix or a permutation of columns). The difficulty lies in the combination of functional and non-functional requirements that are hard to fulfil via the typical approach, *i.e.* writing and deploying complex software.

As stated above, most of the transformations to be written pertain to comparatively few files. As a consequence, one has the following alternatives.

- Perform the transformation fully manually. However, a manual transformation is hard to check. Errors that distort the meaning of the measurement may go undetected and are hard to verify after the fact. Tracking changes needs additional software.
- Write a transformation via a program, be it Python [VD09], R [R 22], or workflow systems like KNIME [Be09]. This has the following potential drawbacks:
  - There is more work needed for testing the code than the work needed to do the transformation itself. This is frustrating, but too little testing may lead to later undetected errors.
  - For a multitude of formats there will be a multitude of pieces of transformation software. Thereby it becomes a challenge to keep track of which input lead to which output using which transformation software.

These problems call for the following:

- A recording-based solution. Creating the transformation software should be as simple as doing the transformation by hand.
- There must be a trace of both source and destination of the transformation. In addition, the software used for transformation should be recorded within the workbook that contains source and transformed data.
- Ideally, the methods used should be platform-independent and should work with as many spreadsheet systems as possible.

In building the solutions, it has to be kept in mind that there are two types of users in most realistic scenarios: (i) *the data steward*, *i.e.* an experienced user whose focus is on data quality. They are typically able to choose their toolchain for performing data transformation. (ii) *the end user*, *i.e.* the scientist who is generating the data and has to provide standards-compliant data. Typically they have the following challenges:

- They are restricted in the tools they can use due to security concerns. The machines are often managed by the institute, which makes it hard to install plugins and add-ons (*e.g.* Excel), install scripts based on languages not yet installed as well as new executables.
- They are restricted in the use of cloud services due to security concerns. Data paths are carefully monitored, and sending early experimental data to a cloud service is discouraged, for example.
- They are restricted in the time they can invest into tools that do not directly increase their chances of getting a paper accepted. It means tools should be easy to use and cannot *e.g.* expect dexterity or an advanced level of long-term concentration for their use.
- Much preliminary data exchange is still done via mail. This favours methods that easily pass antivirus software (*i.e.* no macros). This also favours methods that enable sending related data in one single file as opposed to having to send a collection of files.

ReStoRunT (Record, Store, Run, Trace transformations in Excel sheets) addresses the needs expressed above. It is a recording-based solution that comes in two flavours, (i) an operating protocol that can be performed manually by experimentalists and already captures most of the advantages of ReStoRunT. (ii) A collection of small Python scripts (to be extended) in case the use of Python is possible. We took Python, as it is a frequently used programming language, also in the biological context. In both cases, the original data and the transformed data stay together in one file, enabling easy sending.

Within this paper, we first describe the key properties of MS Excel that play a role later. We then describe a toy example that we solve via ReStoRunT. Afterwards, we describe some software tools that simplify the use of ReStoRunT and then compare the outcome to the state of the art. This is followed by a summary and an outlook on future work.

## 2 Key properties of Spreadsheets and Workbooks

According to Wikipedia [Wi23], the first product introducing the concept of spreadsheets that auto-update was VisiCalc in 1979. It enabled the interactive laying out of data in a table containing cells and combining these cells using formulas. A formula in a cell was able to aggregate information from other cells, such as summing them up. The key innovation was a simple, intuitively graspable way to update the cells when a dependent cell was changed.

However, Lotus 1-2-3, an early successor, advertised already in 1983 that it had functionality for using Lotus sheets as a simple database. Excel, starting in 1985, offered the same. So, it does not come as a surprise that scientists soon took up using Excel as a simple database. And —while database scientists reserve the term database for software that has other properties, such as a well-defined data model— the popularity of spreadsheets as makeshift databases are due to their ease of use and flexibility. The present tool tries to alleviate some of the drawbacks of use of Excel as a database.

### 2.1 Definitions

Within this paper, a spreadsheet  $S$  is viewed as an  $n$ -dimensional arbitrarily large matrix:

$$S = \begin{pmatrix} A1 & B1 & C1 & \dots & Z1 & AA1 & AB1 & \dots \\ A2 & B2 & C2 & \dots & Z2 & AA2 & AB2 & \dots \\ \dots & \dots \\ An & Bn & Cn & \dots & Zn & AAn & ABn & \dots \end{pmatrix} \quad (1)$$

Each element of a spreadsheet is called a cell, and it contains data of an arbitrary type. The format of a cell determines how it is interpreted. Numbers adhering to the locale are recognized as such (*e.g.* 1, 2 in Germany corresponds to 1.2 in the UK). Formulas are expressions that start with an '=' sign.

The cell is uniquely identified by a cell address (or location), which consists of a sheet identification, a column letter, and a row number. For example, cell  $c$  has a cell location  $A6$  in the spreadsheet  $S$ . Formally written as  $S!A6$ , where  $!$  is a delimiter.

Formulas can also reference cells, so  $= A1 * B1$  will be the value obtained by multiplying the number in cell  $A1$  by the number in cell  $B1$ .

Addresses in cells are implicitly relative. So if the formula  $= A1 * B1$  is written into the cell  $C1$ , copying the data from  $C1$  to  $C2$  will change the formula to  $= A2 * B2$ . This is very useful for performing the same operation on numerous cells, *e.g.* multiplying the price by the number of items or similar. Sometimes this is unwanted, *e.g.* when applying the same tax rate to multiple items. For this purpose, it is possible to reference cells fixedly,  $= \$A\$1 * B1$  would become  $= \$A\$1 * B2$  upon copying it to  $C2$ .

In this paper, we propose a *ReStoRunT copy sheet*  $C$  of a source sheet  $S$  with  $n$  lines and  $m$  columns, which is a matrix with cells  $C!Address(l, c)$   $1 \leq l \leq n$ ,  $1 \leq c \leq m$ , where

$Address(l, c)$  denotes the string consisting of column letter and line number for line  $l$  and column  $c$ , and each cell  $C!Address(l, c)$  is a reference to  $S!Address(l, c)$ .

The formula  $= \$A\$1$  references the content of the cell  $A1$ .  $= \alpha!\$A\$1$  denotes the content of the cell  $A1$  in the sheet  $\alpha$ ,  $= \alpha!\$A\$2$  the cell  $A2$ , and so forth. However if  $\alpha!\$A\$2$  is empty,  $= \alpha!\$A\$2$  is not shown and treated as an empty cell, but as 0! This needs to be filtered out, by an if statement, yielding the following formula:

$$= if(\$A\$2 ='''';'''' ; \$A\$2) \tag{2}$$

The copy sheet thus consists of such a formula in each cell.

We call *ReStoRunT transformation sheet* of  $S$  a *ReStoRunT copy sheet* of  $S$  that has been modified, e.g. by moving cells or deleting cells or adding content such as names and labels. A relationship between the source sheet and the ReStoRunT transformation sheet can be viewed as a transformation function applied on the source sheet, returning the ReStoRunT transformation sheet.

Note: Two cells in the ReStoRunT transformation sheet can reference the same cell in the source sheet.

### 3 ReStoRunT by example

Within this section, we will describe ReStoRunT via an example that covers the inner workings of ReStoRunT and give an insight into the outcomes.

#### 3.1 The transformation task

In the scenario, Alice and Bob have agreed on a common format. Denoting values measured for two enzymes (called E1, E2), measured at time points 5, 10, 15, and 20 minutes, and as the measurement can (at times) vary by batch, the batch number is noted. Only Alice needs an average between the experiments for each time point. Each measurement value is accompanied by the possibility to make notes. A hypothetical file may look like the following. The numbers have been picked, of course, so that the reader can easily see how the transformation moves cells:

|           |    |    |         |       |
|-----------|----|----|---------|-------|
| Batch#    | 55 |    |         |       |
| time(min) | E1 | E2 | Average | Notes |
| 5         | 11 | 21 | 16      | Note1 |
| 10        | 12 | 22 | 17      | Note2 |
| 15        | 13 | 23 | 18      | Note3 |
| 20        | 14 | 24 | 19      | Note4 |

Alice needs this data form.

|             |       |       |       |       |
|-------------|-------|-------|-------|-------|
| time (min)  | 5     | 10    | 15    | 20    |
| E1          | 11    | 12    | 13    | 14    |
| E2          | 21    | 22    | 23    | 24    |
|             |       |       |       |       |
| Notes       | Note1 | Note2 | Note3 | Note4 |
| Batch # gel | 55    |       |       |       |

Bob has this data form.

As a consequence, Bob needs to apply transformations in order to share his data with Alice.

### 3.2 Transformation without ReStoRunT

Imagine that it is a one-off exchange of data. Alice needs Bob's data in her format (to feed it into some software or some agreed-on standard format), but currently, there is only one file, and Bob does not want to waste time before knowing there are more files of the same kind.

So, Bob transforms his data manually. In MS Excel, the easiest way to do this is to cut and paste "special", and transpose the data on the way. So he marks data at the *first* line down to the *fifth*, takes an empty sheet and pastes them into that empty sheet, choosing to *transpose* the matrix, to the *second* line of the sheet (Left Table). We replace the empty column with line averages using the formula =AVERAGE(B1:C1) (Right Table).

| time(min) | E1 | E2 |  | Notes |
|-----------|----|----|--|-------|
| 5         | 11 | 21 |  | Note1 |
| 10        | 12 | 22 |  | Note2 |
| 15        | 13 | 23 |  | Note3 |
| 20        | 14 | 24 |  | Note4 |

The transformation of Bob's table.

| time(min) | E1 | E2 | Average | Notes |
|-----------|----|----|---------|-------|
| 5         | 11 | 21 | 16      | Note1 |
| 10        | 12 | 22 | 17      | Note2 |
| 15        | 13 | 23 | 18      | Note3 |
| 20        | 14 | 24 | 19      | Note4 |

We replace the empty column with line averages.

And then we notice that the batch number is missing, which we also have to add on top via two cut-and-paste operations. By this, we have achieved Alice's format.

### 3.3 Weaknesses purely manual transformation

For this one time, this is the most simple that can be done. The goal is met without any overhead. However, if there is any doubt about the accuracy of the transformation ("Did Bob *mispaste*?"), Alice will have to check the original file, which may still be somewhere on Bob's hard disk.

It would have been simpler to verify, had Bob used a script or a computational workflow to modify the data. However, for a one-time transformation spending (in real-life-sized cases) several hours for preparing and testing the script would have been prohibitive.

With the ReStoRunT approach, a couple of minutes of additional work in preparing the sheet will suffice, and the rest will work as before. And the result will be a workbook that (1) contains Bob's original data sheet, and (2) contains a sheet that holds the data in Alice's format. It is a ReStoRunT *transformation sheet*. This sheet contains the information in a

way that (3) enables each cell to trace the origin into Bob’s sheet. And finally, (4) new Bob format sheets can be transformed in the same way into Alice format sheets, reusing the sheet described in (2).

### 3.4 Copy sheet and transformation sheet

In the previous section, we have described our goal. Creating a *transformation sheet* that contains the data in Alice’s format and that can be reused to transform other data in Bob’s format into Alice’s format.

We do so by creating a copy sheet and then modifying it manually.

As described above *ReStoRunT copy sheet*  $\beta$  of  $\alpha$  is a sheet, where each  $c$  in  $\beta$  references the cell in the same line and column in  $\alpha$  using formula 2 in section 2.1.

### 3.5 Creating a transformation sheet by manually applying a sequence of changes

We call Bob’s sheet  $\alpha$ , and its ReStoRunT copy sheet  $\beta$ . In section 3.2, Bob has applied his changes to  $\alpha$ . Now we just apply the same changes to the copy sheet  $\beta$ , instead. The result will just *look* the same. We omit showing the table for brevity.

We have turned the copy sheet into a *transformation sheet* that shows the transformed data, and which —as shown in the next sections— embodies the transformation in a reusable manner.

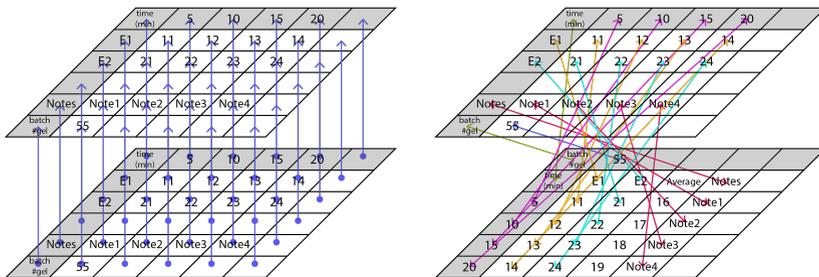


Fig. 1: Right: A ReStoRunT copy sheet referencing the original. Left: A ReStoRunT transformation sheet.

### 3.6 Tracing the result back to the source

Now imagine that the value 12 is in doubt. *We want to know, is there a copy-paste error?* The value 12 is in cell  $\beta!B4$ . Looking into the cell, the cell  $\beta!B$  contains the formula  $=\alpha!$C$2$

whose value is 12, as desired. The headers in the sheet  $\alpha$  appear to be correct. So, after looking at the formula, we can tell where the value of  $\beta!B4$  comes from. The same applies to all other cells in the sheet  $\beta$ .

In other words, the sheet  $\beta$  now contains all references to the original values in  $\alpha$  and it meets Alice's format requirements. The workbook containing both  $\alpha$  and  $\beta$  contains both the original values in the original format as well as the transformation. This also works with single-cell formulas and with many multi-cell formulas.

### 3.7 Rerunning the transformation sheet on a new data sheet

Now, assume that the data exchange between Alice and Bob has been successful. Bob has the sheets  $\alpha$  and the transformation sheet  $\beta$ . He now has additional data  $\alpha'$  that he would like to turn into Alice's format.

All he needs to do is the following two steps:

1. Create a copy  $\beta'$  of  $\beta$  that copies all the formulas. Each formula references a cell in  $\alpha$
2. Now do a replace on all cells in  $\beta'$ : Replace references to  $\alpha$  by references to  $\alpha'$ . This can be done via a simple string replacement on all formulas. Now all these formulas reference the corresponding cells in  $\alpha'$ .

So, by one copy and one search/replace operation, the same transformation was applied to another sheet in Bob's format. This is suitable for small numbers of workbooks and sheets.

## 4 Software supporting ReStoRunT

Creating a copy sheet  $\beta$  for a given sheet  $\alpha$  in a workbook appears to be the most tedious and error-prone step. However, it suffices to create a workbook that contains an empty sheet  $\alpha$  and an  $n \times m$  sheet  $\beta$  that is a copy sheet of the empty sheet  $\alpha$ .  $\beta$  can then be used as a copy sheet for any non-empty sheet of size  $n \times m$  or below.

To further reduce the manual work, e created some lightweight Python tools that simplify using ReStoRunT. We chose Python as a language that is widely accepted and installed. The software is small and open source in order to invite checks by its users. [MM22] contains a repository with the following tools:

```
ReStoRunTify --infile f.xlsx --outfile g.xlsx
```

reads `f.xlsx`, adds ReStoRunT copy sheets for each sheet in `f` and writes the resulting workbook to `g.xlsx`.

```
IsolateReStoRunTsheet --infile f.xlsx \
  --tobeisolated "TestSheet" --outfile isolated.xlsx
```

Takes ReStoRunT-TestSheet from `f.xlsx` and creates a workbook that contains just ReStoRunT-TestSheet and an empty TestSheet. We need the empty TestSheet, as without such a sheet, all the references in ReStoRunT-TestSheet will be broken and replaced by an error string.

```
ApplyReStoRunTsheet --infile f.xlsx --sheetfile g.xlsx \
  --destinationssheet "Sheet 2" --outfile o.xlsx
```

takes the first ReStoRunT sheet in the sheetfile (`g.xlsx`) and applies it to the sheet `--destinationssheet Sheet 2`, and then writes out the resulting workbook to the `--outfile o.xlsx`.

## 5 Advantages and limitations of the ReStoRunT approach

Using simple and well-known means, we have reached a useful way of storing Excel transformations for reuse in small series. These transformations are stored within the workbook and are platform-independent. The representation can be used to create other software for larger series of documents. This is our priority in future work on this topic.

MS Excel has the functionality to trace back formula references to their origins. That makes ReStoRunT more useful, as one can see *visually* which cells depend on which other cells.

ReStoRunT works for arbitrarily large, finite-sized sheets. It is applicable for all use cases where the maximum size of the matrix to be transformed can be determined beforehand. In this paper, we described the transformation as the translation of cells. But also normalisation and other formulas that concern a small, finite number of cells (like the average in our example) are something that is tackled using ReStoRunT.

ReStoRunT uniquely works on the layout of sheets, so far, it does not make use of labels or other content within the sheet.

We see as the main limitations (i) that very large numbers of cells will slow down Excel and (ii) that there are some Excel area functions that make it hard to trace back. For example: Sorting a column will yield results, but it will be hard to trace back which value *really* was the third biggest value in a cell set containing 30.000 cells. This can be countered by cascading ReStoRunT sheets thereby extending the detail of traces of changes.

ReStoRunT is using basic Excel formulas, RStoRunT works on Google Sheets, LibreOffice, as well as Apple Numbers and Gnumeric.

## 6 State of the art compared to ReStoRunT

For re-applying changes, MS Excel has a built-in macro recorder. When using it, it is hard to build working code without modifying the recorded VBA macro afterwards. This is problematic, as the recording needs to be done by experimental scientists who cannot practise this task sufficiently to reach proficiency. Also, the intended users cannot be expected to be fluent in VBA. In addition, a drawback of recorded Excel macros is that many spam filters mark .xlsm (Excel with Macro) files as SPAM, as the powerful embedded VBA code poses a security risk. Furthermore, the receiver is asked if they want to run the security risk of using the macro. Especially novices will not be equipped to take this decision. In addition to that, the resulting transformation code is platform dependent, as it is expressed in Visual Basic for Applications (VBA). Only through an analysis of a given macro a proficient reader will be able to find out what field was the source of a change. The reader will have to *invert* the operations done while recording the macros to find the source of data.

In contrast, one click on the formula in a ReStoRunT sheet will show which cells in the original data sheet contributed to the current value. Furthermore, ReStoRunT needs only key Excel mechanisms for functioning that function across a wide range of spreadsheet tools, as stated above.

*InSituTrac* [As13] is a comprehensive Excel add-in for recording changes to Excel files. The purposes are tracing provenance and re-applying changes. The comprehensive solution features visualising types and sequences of changes to Excel sheets. Functionality-wise it goes far beyond ReStoRunT. The Excel add-in centres around a ribbon in Excel that gives access to the recording and exploration functionality.

In contrast, ReStoRunT is cross-platform, packages both original and result in one workbook, and the provenance information can be perused without resorting to any add-in.

Google Sheets [Go] are a cloud service for spreadsheets that provide macro recording and change tracking information. However, recorded macros are not exported alongside an Excel export of Google Sheets. So the relation between Sheet and Macro is lost. Copying workbooks provide a way to get a script into another workbook. However, again the users are asked to take uncomfortable security decisions.

Exemplify [Sh13] was a tool for doing traceable changes to Excel files in the frame of Immunoblot experiments. However, this was a large piece of configurable bespoke software with the problems we described above, *i.e.* it needed too much configuration work for each format change.

openRefine [te22] is a tool built for cleaning dirty data. It is a separate application to be installed in user space. It provides functionality to import sheets and then modify them using point 'n' click as well as multi-cell operations. The traces of such modifications can be recorded, stored, imported, and reused. However, the transformation is not shipped with

the data, and openRefine first imports data into one structure, and then re-exports them. This makes it hard to work with complex workbooks.

Workflow tools such as KNIME [Be09] and Galaxy [Af18] provide rich table functionality. But just as using Python's Pandas library [te20], R [R 22], R tidyverse [Wi19], and another tooling, creating transformers in these tools does not happen by simple recording and needs to be tested to a greater extent than recording based solutions. Some of them also import the table into an intermediate format, thus losing the formatting information of the initial table.

[Wo11] is a tool for adding ontology information to spreadsheets, a complementary approach. It can read workbooks, add hidden sheets with ontology information and then store the sheet. These data can then subsequently be read by other tools. RightField's use is complementary to the tools described above.

To our knowledge, ReStoRunT has its use in the space of Spreadsheet-related tools, being a useful addition because it is simple, not in the cloud, and doing quality control using a ReStoRunT sheet does not need anything beyond standard software.

## 7 Conclusion

We argued that transforming Excel files and similar spreadsheets is an important task in experimental biological work. The difficulty lies in the fact that one needs many *different* transformations that need to be *traced* and possibly *rerun* several times, but not rerun often enough to warrant a large development or configuration effort.

For this task, we have proposed ReStoRunT, *i.e.* recording and storing transformations such that results can be traced to their origin and finally can be rerun, *i.e.* applied to new data.

ReStoRunT can be used entirely manually as a set of Excel practises or complemented via tooling, of which we present an initial version. We hope to help scientists in sharing their experimental data in a harmonized manner.

## Acknowledgements

Müller and Mertová are funded by HITS and the Klaus Tschira Foundation, KTS. Müller had been co-funded by MESI-STRAT. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 754688. Much of this work came from discussions with *Ines Heiland* and *Cecilia Barile* about their needs for Excel transformations and traceability. Another input came from programming work that Mertová and Müller did for ASSR, the Samaritans of Slovakia. We thank Stefan Giuliani of ASSR for a guided tour of their needs and their application in real life.

## References

- [Af18] Afgan, E.; Baker, D.; Batut, B.; van den Beek, M.; Bouvier, D.; Čech, M.; Chilton, J.; Clements, D.; Coraor, N.; Grüning, B. A.; Guerler, A.; Hillman-Jackson, J.; Hiltemann, S.; Jalili, V.; Rasche, H.; Soranzo, N.; Goecks, J.; Taylor, J.; Nekrutenko, A.; Blankenberg, D.: The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Res.* 46/W1, W537–W544, 2018.
- [As13] Asuncion, H. U.: Automated data provenance capture in spreadsheets, with case studies. *Future Generation Computer Systems* 29/8, Including Special sections: Advanced Cloud Monitoring Systems & The fourth IEEE International Conference on e-Science 2011 — e-Science Applications and Tools & Cluster, Grid, and Cloud Computing, pp. 2169–2181, 2013, ISSN: 0167-739X, URL: <https://www.sciencedirect.com/science/article/pii/S0167739X13000691>.
- [Be09] Berthold, M. R.; Cebon, N.; Dill, F.; Gabriel, T. R.; Kötter, T.; Meinel, T.; Ohl, P.; Thiel, K.; Wiswedel, B.: KNIME - the Konstanz Information Miner: Version 2.0 and Beyond. *SIGKDD Explor. Newsl.* 11/1, pp. 26–31, Nov. 2009, ISSN: 1931-0145, URL: <http://doi.acm.org/10.1145/1656274.1656280>.
- [CK04] Carroll, J.; Klyne, G.: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, <https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, W3C, Feb. 2004.
- [FA22] FAIRsharing.org: MIBBI; Minimum Information for Biological and Biomedical Investigations, <https://fairsharing.org/3518>, [Online, accessed 2022-12-02], 2022.
- [Go] Google Workspace, G.: Google Sheets: Online Spreadsheet Editor, URL: <https://www.google.com/sheets/about/>.
- [Hu03] Hucka, M.; Finney, A.; Sauro, H. M.; Bolouri, H.; Doyle, J. C.; Kitano, H.; Arkin, A. P.; Bornstein, B. J.; Bray, D.; Cornish-Bowden, A.; Cuellar, A. A.; Dronov, S.; Gilles, E. D.; Ginkel, M.; Gor, V.; Goryanin, I. I.; Hedley, W. J.; Hodgman, T. C.; Hofmeyr, J.-H.; Hunter, P. J.; Juty, N. S.; Kasberger, J. L.; Kremling, A.; Kummer, U.; Novère, N. L.; Loew, L. M.; Lucio, D.; Mendes, P.; Minch, E.; Mjolsness, E. D.; Nakayama, Y.; Nelson, M. R.; Nielsen, P. F.; Saku-rada, T.; Schaff, J. C.; Shapiro, B. E.; Shimizu, T. S.; Spence, H. D.; Stelling, J.; Takahashi, K.; Tomita, M.; Wagner, J.; Wang, J.; Forum, S. B. M. L.: The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19/4, pp. 524–531, Mar. 2003.
- [MM22] Mueller, W.; Mertová, L.: ReStoRunT GitHub repository, <https://github.com/mertova/ReStoRunT>, [Online, accessed 2022-12-01], 2022.
- [R 22] R Core Team: R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2022, URL: <https://www.R-project.org/>.

- [Sh13] Shi, L.; Jong, L.; Wittig, U.; Lucarelli, P.; Stepath, M.; Mueller, S.; D'Alessandro, L.; Klingmüller, U.; Müller, W.: Excmplify: a flexible template based solution, parsing and managing data in spreadsheets for experimentalists. *J Integr Bioinform.* 2/10, p. 220, Apr. 2013.
- [te20] pandas development team, T.: pandas-dev/pandas: Pandas, version latest, Feb. 2020, URL: <https://doi.org/10.5281/zenodo.3509134>.
- [te22] openRefine team: openRefine, <http://openrefine.org/>, [Online, accessed 2022-12-01], 2022.
- [VD09] Van Rossum, G.; Drake, F. L.: Python 3 Reference Manual. CreateSpace, Scotts Valley, CA, 2009, ISBN: 1441412697.
- [Wi16] Wilkinson, M. D.; Dumontier, M.; Aalbersberg, I. J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.-W.; da Silva Santos, L. B.; Bourne, P. E., et al.: The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3/, 2016.
- [Wi19] Wickham, H.; Averick, M.; Bryan, J.; Chang, W.; McGowan, L. D.; François, R.; Grolemund, G.; Hayes, A.; Henry, L.; Hester, J.; Kuhn, M.; Pedersen, T. L.; Miller, E.; Bache, S. M.; Müller, K.; Ooms, J.; Robinson, D.; Seidel, D. P.; Spinu, V.; Takahashi, K.; Vaughan, D.; Wilke, C.; Woo, K.; Yutani, H.: Welcome to the tidyverse. *Journal of Open Source Software* 4/43, p. 1686, 2019.
- [Wi23] Wikipedia: VisiCalc, 2023, URL: <https://en.wikipedia.org/wiki/VisiCalc>, visited on: 01/22/2023.
- [Wo11] Wolstencroft, K.; Owen, S.; Horridge, M.; Krebs, O.; Mueller, W.; Snoep, J.; du Preez, F.; C., G.: RightField: embedding ontology annotation in spreadsheets. *Bioinformatics* 14/27, pp. 2021–2, July 2011.



# A Core Ontology to Support Agricultural Data Interoperability

Aly Abdelmageed<sup>1</sup>, Shahenda Hatem<sup>1</sup>, Tasneem Wael<sup>1</sup>, Walaa Medhat<sup>1</sup>, Birgitta König-Ries<sup>2</sup>, Susan F. Ellakwa<sup>3</sup>, Passent Elkafrawy<sup>1,4</sup>, Alsayed Algergawy<sup>2,5</sup>

**Abstract:** The amount and variety of raw data generated in the agriculture sector from numerous sources, including soil sensors and local weather stations, are proliferating. However, these raw data in themselves are meaningless and isolated and, therefore, may offer little value to the farmer. Data usefulness is determined by its context and meaning and by how it is interoperable with data from other sources. Semantic web technology can provide context and meaning to data and its aggregation by providing standard data interchange formats and description languages. In this paper, we introduce the design and overall description of a core ontology that facilitates the process of data interoperability in the agricultural domain.

**Keywords:** Semantic Web; Ontology; Knowledge Modeling; Agriculture

## 1 Introduction

The Agricultural Research Center (ARC)<sup>6</sup> and the Central Lab for Agricultural Expert Systems (CLAES)<sup>7</sup> have been established to enhance the productivity of knowledge engineers in building agricultural expert systems in Egypt. The ARC center has several institutes focusing on soil, water, environment, and field crops. The outcome of those institutes is a large amount of scattered and not well-described data, which makes it hard to integrate and reuse. Furthermore, there is no availability of those data to the international community due to lack of data representation and standardization. For example, the Registry of Research Data Repositories<sup>8</sup> has the resource of information about research data repositories, including agricultural data. Even though it indexes and provides extensive information about more than 270 agricultural data repositories. However, there are no records from Egypt. Therefore, there is a growing need to start a process that supports the interoperability of agricultural data in Egypt.

---

<sup>1</sup> Information Technology and Computer Science School, Nile University, Egypt

<sup>2</sup> Heinz Nixdorf Chair for Distributed Information Systems, Friedrich-Schiller University of Jena, Germany

<sup>3</sup> Climate Change Information Center Renewable Energy Expert Systems, Agricultural Research Center, Egypt

<sup>4</sup> Collage of Engineering, Effat University, SA

<sup>5</sup> alsayed.algergawy@uni-jena.de

<sup>6</sup> <http://www.arc.sci.eg/>

<sup>7</sup> <http://www.claes.sci.eg/>

<sup>8</sup> [https://www.re3data.org/\(http://doi.org/10.17616/R3KG8X](https://www.re3data.org/(http://doi.org/10.17616/R3KG8X) last accessed: 2022-11-27)

The objective of this work is to enhance and improve the interoperability of agricultural data in Egypt collected and coordinated from CLAES. In this context, the semantic web, in general, and ontology, in particular, play a crucial role. As the ontology formally represents key concepts, properties, relationships, and axioms of a given domain, where it allows a richer set of relationships and constraints among key terms in the domain [Pr13, Jo16, Dr19]. These silent features allow the ontology to make domain-specific knowledge more explicit and in a machine-readable format. As it is hard to develop a single ontology that covers the entire scope of the agricultural domain, we initially focus on developing a core ontology for key terms extracted from different datasets collected at CLAES. To this end, we design and develop a semantic model to capture the domain knowledge and to be used as the core component in carrying out the Egyptian agricultural data interoperability. It is not a simple task since agricultural data are described using different languages.

Furthermore, terminologies used in agricultural data, such as names of the crop, equipment, and activity, have not been standardized because agriculture is local [Jo16]. To this end, we exploit available resources at CLAES, such as datasets, technical reports, and surveys, to extract and collect main terms relevant to the agricultural domain. To develop a core ontology from the extracted set of terms, we employ the fusion-merge approach [PM04, OYD21, Sc11], where these extracted terms are then used to localize related ontologies that can be reused as a basis for the core ontology design from available ontology portals, such as BioPortal<sup>9</sup> and AgroPortal<sup>10</sup>. We employed module extractor strategy to the selected set of ontologies to reduce the number of selected concepts and properties and to ensure that the core ontology will not contain unneeded concepts making it more complex than necessary [Al20, Br22]. These modules are then combined to form the initial version of the core ontology. Further improvements are made, such as revising the ontology and adding missing concepts. The role of domain experts is very significant and essential in almost all steps during the development of the core ontology.

The rest of the paper is organized as follows: In the next section, we present the background and related work. The main methodology for developing the core ontology will be introduced in Section 3. Section 4 is devoted to concluding the paper and discussing the open issues and future work.

## 2 Related work

The rapidly growing population and climate changes have been accompanied by the emergence of new priorities in agricultural research, which is marked by a large volume and heterogeneous range of data sources and formats [De23, Dr19]. This data in itself is meaningless and isolated and therefore may offer little value to the farmer. The usefulness of data comes from context and meaning, as well as its aggregation with other data sources.

---

<sup>9</sup> <https://biportal.bioontology.org/>

<sup>10</sup> <http://agroportal.lirmm.fr/>

Semantic web technology can provide context and meaning to data and its aggregation by providing standard data interchange formats and data description languages. The RDA working group, 'Agrisemantics WG'<sup>11</sup>, has been working on gathering community-based requirements and use cases for an infrastructure that supports the use of semantics for agricultural data interoperability. However, the working group has been criticized for not focusing on data from non-EU countries and for not addressing core ontologies enough. Despite the large number of domain-specific ontologies in the field of agriculture, there is a lack of core ontologies that link foundational and domain-specific ontologies.

### 3 Methodology

In this section, we focus on the strategy used to develop a core ontology that will be used as a seed for the interoperability of agricultural data. In this context, we first will introduce the use case and scenario where the development of such a core ontology is necessary, then we will present our methods to achieve this goal, and finally, we will describe the main outcome of the development process.

**Scenario.** To motivate the presented work, we shall start introducing the current scenario of representing and managing agricultural data in Egypt. As mentioned, the Agricultural Research Center (ARC) is the responsible unit for gathering agricultural data. These data can be provided to the center in different formats. It could be unstructured data, such as technical reports by scientists and surveys through interviews with farmers. It could also be semi-structured data in XML formats or/and tabular data. The results of modeling and representing these different data sources are a number of isolated XML files (datasets). For example, as shown in Fig. 1, where a piece of three different datasets about Wheat, Rice, and Tomato are illustrated. The figure shows that the three different datasets represent the same concept 'Soil Salinity'. It has to be repeated in each dataset with a different level of knowledge. For example, the Wheat dataset models the 'Soil Salinity' with more information, such as it gives the Arabic label of the concept, which is missing information in the other two datasets. Furthermore, the Wheat and Rice datasets provide a legal value 'Salinity' for the concept, while it is not provided in the Tomato dataset. The legal value is in both English and Arabic languages in the Wheat dataset, while it is specified only in English in the Rice dataset.

Another example that demonstrates the need to unify and organize the same piece of information in the same way is illustrated in Fig. 2. The figure shows that the concept 'Agricultural Operations' is defined in the Wheat and Rice with three subclasses: 'Pre-Cultivation', 'Cultivation', and 'Post-Cultivation' with different level of information. The concept 'Pre-cultivation' from the Rice dataset has one property defined only in English, while it has two properties defined in English and Arabic in the Wheat dataset. These examples show that there is a large redundancy and different

<sup>11</sup> <https://www.rd-alliance.org/groups/agrisemantics-wg.html>

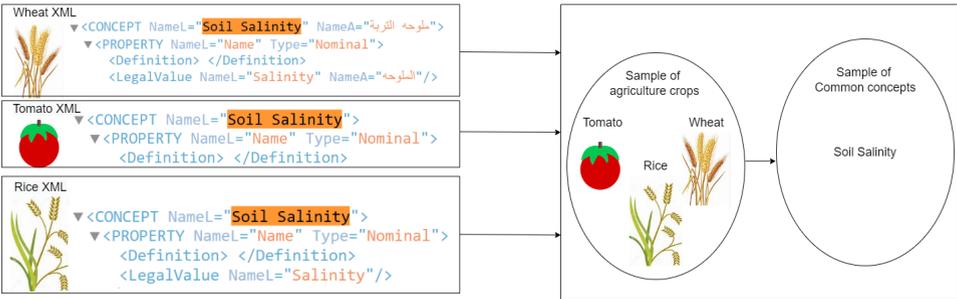


Abb. 1: Data sources challenges: Example 1

representations of the same concepts. Therefore, there is a growing need to unify the representation of these concepts across different datasets, which motivates the development of a core ontology.

As shown in Figures 1 and 2, these words are repeated in these XML files with the same definitions; thus, domain experts select one of them with its children. In order to have the semantic model has no redundant concepts. Some concepts have the same meaning but different structures like disease and disorder, hence domain experts select one of them. All data types will be linked together using semantic modeling approaches to make it reusable. Providing a global ontology for the Egyptian community while having wide exposure to the international research of foreign countries. The international agricultural knowledge base will provide a common understanding of the huge and continuously growing domain of agricultural knowledge. The idea is to add Arabic plant knowledge to the global semantic agricultural knowledge with the German counterpart for a World Global plant knowledge base. This big goal is to be achieved after defining a core concept design for aligning and merging CLAES ontologies for a global agricultural knowledge base. Such structure will facilitate semantic definition and integration of a huge amount of knowledge with different criteria and presentation.

**Strategy.** The entire process of developing a core ontology is illustrated in Fig. 3, where the proposed framework is illustrated. The figure shows that the framework has four main layers. The data sources layer keeps track of data resources used during the development of the core ontology. As shown in Fig. 3, these data resources include unstructured data (PDF files representing technical reports and/or surveys) and semi-structured data (XML datasets). The preprocessing layer is to allow reading of different data resources. For that, it has a translator component to translate Arabic into English. Furthermore, to enable the processing of PDF files, we transform these files into text format using Python libraries such as `pdftextract`<sup>12</sup>. We decided to use these libraries as it is a very simple and efficient python PDF text extractor that uses the `xpdf c++` library. It allows the extraction of text from the

<sup>12</sup> <https://pypi.org/project/pdftextract/>

|  |   |
|--|---|
| <pre> &lt;CONCEPT Name="Agricultural Operations"&gt;   &lt;SubConceptOf /&gt;   &lt;Definition /&gt; &lt;/CONCEPT&gt; &lt;CONCEPT Name="Pre-Cultivation"&gt;   &lt;PROPERTY Name="Time" Type="String"&gt;     &lt;Definition&gt;Number of days before cultivation &lt;/Definition&gt;   &lt;/PROPERTY&gt;   &lt;SubConceptOf Agricultural Operations &lt;/SubConceptOf&gt;   &lt;Definition /&gt; &lt;/CONCEPT&gt; &lt;CONCEPT Name="Cultivation"&gt;   &lt;SubConceptOf Agricultural Operations &lt;/SubConceptOf&gt;   &lt;Definition /&gt; &lt;/CONCEPT&gt; &lt;CONCEPT Name="Post-Cultivation"&gt;   &lt;SubConceptOf Agricultural Operations &lt;/SubConceptOf&gt;   &lt;Definition /&gt; &lt;/CONCEPT&gt; </pre> | <pre> &lt;CONCEPT Name="Agricultural Operations" NameA="العمليات الزراعية"&gt;   &lt;SubConceptOf /&gt;   &lt;Definition&gt;A set of operations that being applied during the agricultural process and is   &lt;DefinitionA&gt;مجموعة عمليات يتم انباء الزراعة. مجموعة عمليات يتم بعد الزراعة.   &lt;/CONCEPT&gt; &lt;CONCEPT Name="Pre-Cultivation" NameA="قبل الزراعة"&gt;   &lt;PROPERTY Name="Time" Type="String" NameA="الوقت" Prompt="" PromptA=""&gt;     &lt;Definition&gt;Number of days before cultivation &lt;/Definition&gt;     &lt;DefinitionA&gt;عدد الأيام التي سبق الزراعة&lt;/DefinitionA&gt;   &lt;/PROPERTY&gt;   &lt;SubConceptOf Agricultural Operations &lt;/SubConceptOf&gt;   &lt;Definition&gt;operations occurred before the agricultural operations &lt;/Definition&gt;   &lt;DefinitionA&gt;عمليات يتم قبل الزراعة &lt;/DefinitionA&gt; &lt;/CONCEPT&gt; &lt;CONCEPT Name="Cultivation" NameA="إنشاء الزراعة"&gt;   &lt;SubConceptOf Agricultural Operations &lt;/SubConceptOf&gt;   &lt;Definition&gt;operations occurred during the agricultural operations &lt;/Definition&gt;   &lt;DefinitionA&gt;عمليات يتم أثناء العملة الزراعية &lt;/DefinitionA&gt; &lt;/CONCEPT&gt; &lt;CONCEPT Name="Post-Cultivation" NameA="بعد الزراعة"&gt;   &lt;SubConceptOf Agricultural Operations &lt;/SubConceptOf&gt;   &lt;Definition&gt;operations occurred after the agricultural operations &lt;/Definition&gt;   &lt;DefinitionA&gt;عمليات يتم بعد الزراعة &lt;/DefinitionA&gt; </pre> |
| Rice dataset   | Wheat dataset   |

Abb. 2: Data sources challenges: Example 2

whole PDF or a specific page from the PDF file. Another important component is the XML reader which parses XML files to extract elements and their properties. For example, the XML Reader component reads the Wheat dataset, shown in Fig. 2 and extracts the element 'Agricultural Operation' with a definition 'A set of operations that is applied during the agricultural process and is divide according to the applying time to 1-PreCultivation 2-During Cultivation 3-Post Cultivation' as well as the associated definition in Arabic. The set of extracted texts is preprocessed and input to a natural language processing API. At that point, we make use of the TextRazor demo version<sup>13</sup>. It is an easy-to-use API and very effective. The main goal to use the TextRazor tool is to extract main terms (entities) from text information, such as element names, and comments. For example, input the above definition to the NLP API results in the following terms: set, operation, agricultural process, cultivation. As shown in Fig. 3, the role of domain experts is needed at the end of the preprocessing step to validate and confirm the set of extracted terms. At least three experts from CLAES have proved the preprocessing outcome.

Once we have the set of relevant terms, the next step, as shown in Fig. 3, is to start the semantic modeling process. The semantic modeling layer has three main components: ontology selection, module extractor, and ontology merging. The first component, ontology selection, is to select a set of relevant ontologies that cover the set of input terms. To this end, we make use of the available ontology portals, such as BioPortal and AgroPortal. To this end, we make use of available APIs from the two portals, which accept the set of terms as input and return back a set of ontologies that may cover the terms. We extracted important information from these ontologies to be revised by domain experts to select the most suitable ontologies for the domain. For example, the term 'Fruit' has been found in 24 ontologies representing different pieces of the domain. The fruit concept is defined in the Agricultural Growers Resource Organization (AGRO)<sup>14</sup> as a multi-tissue plant structure

<sup>13</sup> <https://www.textrazor.com/>

<sup>14</sup> <https://bioportal.bioontology.org/ontologies/AGRO>

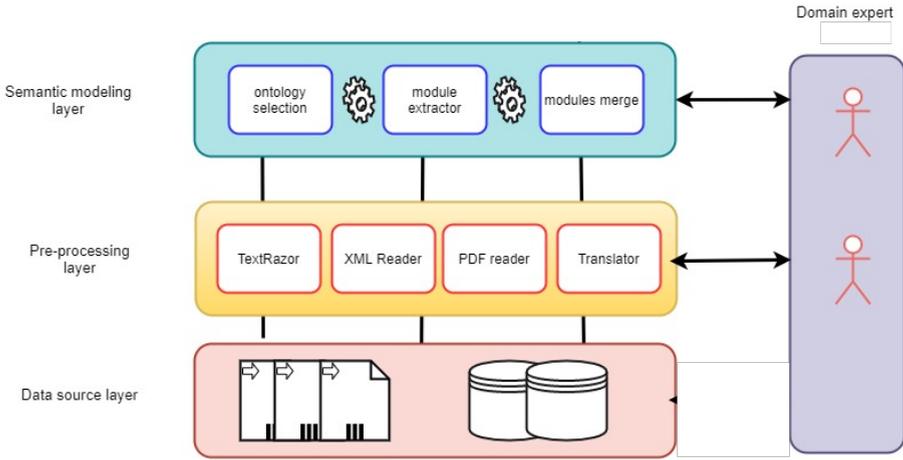


Abb. 3: Proposed framework architecture

that develops from a Gynoecium or a single carpel, and at maturity may have as parts one or more seeds. Also, it may contain additional plant structures that are part of a flower and mature along with the Gynoecium, such as a receptacle. A fruit may develop without fertilization in cases of parthenocarpy, apomixis, or other hormone-induced conditions and may not always contain seeds. When annotating fruit that is referred to as ‘aggregate’, ‘multiple’, or ‘compound’, it is annotated directly to the appropriate plant structure, such as receptacle, hypanthium or infructescence. Fruits only occur in angiosperms. While it is defined within the Medical Subject Headings (MESH)<sup>15</sup> as the fleshy or dry ripened ovary of a plant, enclosing the seed or seeds. Another example to demonstrate the need for feedback from domain experts is the term 'crop' exists in 16 different ontologies, but our experts selected only two definitions that align with the intended meaning.

After having a set of ontologies, for each term we extracted the set of corresponding concepts from different ontologies along with their URIs, labels, and definitions (if they exist). Then the domain experts shall validate the extracted concepts. After settling on a number of ontologies to be adopted according to the fusion/merge strategy [AKR19], a module extractor is applied to each ontology to elicit smaller partitions from the selected set of ontologies. Those concepts are the ones containing only relevant concepts and those needed to connect them. Finally, these sets of partitions were combined and merged to form the initial version of the new ontology.

**Outcomes.** The outcome of applying the proposed approach to available data sources at CLAES is summarized in Table 1, and some of the core concepts and their initial

<sup>15</sup> <https://bioportal.bioontology.org/ontologies/MESH>

relationships are shown in Fig. 4. The table shows that after applying the preprocessing step, we got 211 unique terms extracted from available data sources at CLAES. Using these terms to look up similar concepts from BioPortal and AgroPortal we found 178 concepts. After applying the module extractor and filler irrelevant concepts we got 147 concepts from 11 ontologies. A first trial to build the core ontology is shown in Fig. 4. The core concepts, as well as the related concepts, form the basis for semantically modeling significant assumptions in the Egyptian agricultural sector.

Tab. 1: Data Description Results

|   |                                    |
|---|------------------------------------|
| Number of extracted terms from available resources    | 211                                |
| Number of concepts found in relevant ontology portals | 178                                |
| Size of output of extracted concepts from the portal  | 18324                              |
| Number of the concepts after filtering ontologies     | 147 which belongs to 11 ontologies |

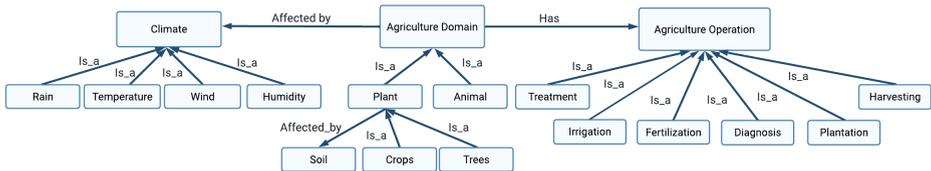


Abb. 4: Extracted concept

All the resources related to the design of the core ontology as well as the first versions of the ontology are accessible online at <https://github.com/fusion-jena/agriSem>.

## 4 Conclusions and Future work

We attempt to address the interoperability of agricultural data in Egypt. We draw the first step towards achieving this goal by developing a core ontology that covers core concepts in the domain. Therefore, we proposed a framework to achieve this goal by producing one merged ontology of core concepts. Even some successful steps have been drawn to reach the main goal; however, several rooms for improvement and future directions are arising. The first is to enhance this initial version of the core ontology, as more work is needed to edit, revise and consider domain experts’ feedback before publishing and/or deployment of the ontology. This results into the next issue of how to involve domain experts in the development process. It is a very hard and time-consuming process, where intelligent solutions have to be proposed.

## Acknowledgments

This work has been funded by the German Academic Exchange Service (DAAD) and Standards and Trade Development Facility (STDF) as part of Semantic Web Technologies for Agricultural data interoperability (AgriSem).

## Literaturverzeichnis

- [AKR19] Algergawy, Alsayed; König-Ries, Birgitta: Partitioning of BioPortal Ontologies: An Empirical Study. In: SWAT4HCLS. S. 84–93, 2019.
- [Al20] Algergawy, Alsayed; Babalou, Samira; Klan, Friederike; König-Ries, Birgitta: Ontology Modularization with OAPT. *Journal on Data Semantics*, 9(2-3):53–83, 2020.
- [Br22] Bravo, Maricela; González-Villarreal, Darinel; Reyes-Ortiz, José A; Sánchez-Martínez, Leonardo D: Modularization Method to Reuse Medical Knowledge Graphs. *Applied Sciences*, 12(22):11816, 2022.
- [De23] Devare, Medha; Arnaud, Elizabeth; Antezana, Erick; King, Brian: Governing agricultural data: Challenges and recommendations. *Towards Responsible Plant Data Linkage: Data Challenges for Agricultural Research and Development*, S. 201, 2023.
- [Dr19] Drury, Brett; Fernandes, Robson; Moura, Maria-Fernanda; de Andrade Lopes, Alneu: A survey of semantic web technology for agriculture. *Information Processing in Agriculture*, 6(4):487–501, 2019.
- [Jo16] Joo, Sungmin; Koide, Seiji; Takeda, Hideaki; Horyu, Daisuke; Takezaki, Akane; Yoshida, Tomokazu: Agriculture Activity Ontology: An Ontology for Core Vocabulary of Agriculture Activity. In: *ISWC (Posters & Demos)*. 2016.
- [OYD21] Osman, Inès; Yahia, Sadok Ben; Diallo, Gayo: Ontology integration: approaches and challenging issues. *Information Fusion*, 71:38–63, 2021.
- [PM04] Pinto, Helena Sofia; Martins, João P: Ontologies: How can they be built? *Knowledge and information systems*, 6(4):441–464, 2004.
- [Pr13] Prestes, Edson; Carbonera, Joel Luis; Fiorini, Sandro Rama; Jorge, Vitor AM; Abel, Mara; Madhavan, Raj; Locoro, Angela; Goncalves, Paulo; Barreto, Marcos E; Habib, Maki et al.: Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.
- [Sc11] Scherp, Ansgar; Saathoff, Carsten; Franz, Thomas; Staab, Steffen: Designing core ontologies. *Applied Ontology*, 6(3):177–221, 2011.

# The InsightsNet Climate Change Corpus (ICCC) -

## Compiling a Multimodal Corpus of Discourses in a Multi-Disciplinary Domain

Elena Volkanovska,<sup>1</sup> Sherry Tan,<sup>2</sup> Changxu Duan,<sup>3</sup> Sabine Bartsch<sup>4</sup> and Wolfgang Stille<sup>5</sup>

**Abstract:** The discourse on climate change has become a centerpiece of public debate, thereby creating a pressing need to analyze the multitude of communications created by the participants in this communication process. In addition to text, information on this topic is communicated multimodally, through images, videos, tables and other data objects that are embedded within documents and accompany the text. This paper presents the process of building a multimodal pilot corpus to the InsightsNet Climate Change Corpus (ICCC) using natural language processing (NLP) tools to enrich corpus metadata, thus building a dataset that lends itself to the exploration of the interplay between the various modalities that constitute the discourse on climate change.

**Keywords:** corpus; climate change; computational linguistics; annotation; metadata

## 1 Introduction

In recent years, the topic of climate change has taken center stage in discourses across different segments of society through different channels, media and publications. While climate scientists are in agreement that climate change is ongoing and real, debates on this topic as well as its influences on policy-makers remain highly controversial [SP21].

With the surge of published data on the topic of climate change, linguistics as well as other related disciplines have identified the study of data representing discourses on climate change as a research desiderate in order to gain a better understanding of this multidisciplinary field and the role played by a diverse set of participants with different scientific and political backgrounds who are assuming different roles and interests. In order to enable such studies, research is needed to collect and organize suitable corpora in a comprehensive and

---

<sup>1</sup> Technische Universität Darmstadt, Corpus and Computational Linguistics, Residenzschloss 1, 64283 Darmstadt, Deutschland elena.volkanovska@tu-darmstadt.de

<sup>2</sup> Technische Universität Darmstadt, Corpus and Computational Linguistics, Residenzschloss 1, 64283 Darmstadt, Deutschland sherry.tan@tu-darmstadt.de

<sup>3</sup> Technische Universität Darmstadt, Corpus and Computational Linguistics, Residenzschloss 1, 64283 Darmstadt, Deutschland changxu.duan@tu-darmstadt.de

<sup>4</sup> Technische Universität Darmstadt, Corpus and Computational Linguistics, Residenzschloss 1, 64283 Darmstadt, Deutschland sabine.bartsch@tu-darmstadt.de

<sup>5</sup> Technische Universität Darmstadt and Hessian Center for Artificial Intelligence (hessian.AI) wolfgang.stille@hessian.ai

meaningful way to inform the different communities engaging and interested in relevant discourses as well as processes concomitant with their roles as scientists, laypersons, politicians, managers and many others involved in the relevant debates and policy making processes. According to [LCJ20], the climate change related topic of global warming “has received little attention in natural language processing [NLP] despite its real world urgency”. One plausible reason for this may be attributed to the lack of available corpora focusing on climate change. Additionally, as a topic - like many topics with a multidisciplinary coverage - climate change is represented in many publications not merely by means of natural language text, but also by means of a multitude of modalities such as images, maps, data tables and visualizations that are hardly captured, let alone systematically analysed for their contribution at all. So while there may be an abundant volume of digital text to be potentially included in corpora, the demonstration of textual and embedded multimodal data objects extracted and stored together in a corpus on the topic of climate change is still lacking. Therefore, the research reported in this paper aims to fill this gap by building multimodal corpora representing discourses from the domain of climate change across different genres. We furthermore set out to demonstrate some exemplary methods from corpus and computational linguistics to enrich the corpus data by metadata and annotations to allow for more in-depth analyses to further our understanding of discourses on the topic of climate change. We believe the analysis of such corpora and the study of the interlinking between the multimodal objects with its textual counterparts will create new insights into the topic of climate change and drive new discussions across various communities.

## **2 Overview of corpora for discourse analysis on climate change**

Prior to embarking on corpus-building, we explored existing corpora and datasets that have been used in previous studies on the climate change discourse. A good overview of datasets used to investigate the debate on climate change by practitioners in the community of NLP and social sciences is provided in [SP21]; unfortunately, none of these studies takes multimodality into account. A further potentially relevant climate change dataset is the Science Daily Climate Change (SciDCC) dataset, presented in [MM21], which includes approximately 11,000 news articles scraped on the topics “Earth and Climate” and “Plant and Animals” of the Science Daily website. Yet, this is a text-only resource as well. There is a limited number of studies on the topic of climate change conducted on multimodal corpora, but these are largely combinations of texts and photographic illustrations (see [ADY11] and [We16]).

The exploration of existing corpora on the topic of climate change revealed that while they are well-suited for text-based discourse analysis, none of them can help us fully address the objective of our study, which is to analyse the climate change discourse as an interaction between various modalities. The corpora that we inspected do not store data objects of different formats in a single corpus in a manner that lends itself to the study of the interplay between a document’s text and any multimedia content embedded in it. In addition, existing

multimodal corpora take into consideration a set number of media types, which does not allow for the exploration of the range of embedded media types. Rather than moulding our research to fit the data that was readily available at the time this study began, we decided to build a multimodal corpus from authentic data that would allow us to examine (1) the type of modalities embedded in a document, and (2) how different modalities contribute to the discourse on climate change.

### **3 Developing a pilot corpus**

The pilot corpus described in this section is a precursor to ICCC. The objective is to explore the possibilities of developing a multimodal corpus on climate change and to systematically learn more about the challenges before expanding it. At the onset of the corpus-building process for the pilot corpus two main criteria were devised: the corpus had to contain content in both English and German, and any collected multimedia content had to be embedded in the document. We refrain from incorporating stand-alone collections of single-modality data such as collections of images or photos etc. We did not set a limit on the types of multimodal data to be collected with the expectation that we will encounter data objects beyond images and videos. Beyond this, we adhere to a fairly standard corpus-design procedure, which includes the following steps: (1) identify genres of interest and data sources that contain suitable content; (2) contact copyright holders to obtain their approval to collect and use the data; (3) define metadata properties to store relevant information; (4) collect the data from each data source, (5) parse it in a project-specific corpus structure.

#### **3.1 Identifying genres, data sources, and obtaining copyright permissions**

The objective in this step was to ensure that each genre included in the corpus represents various entities or members of society that actively take part in the public discourse on climate change. The pilot corpus entails content from three genres: academic papers on climate change, reports published by the International Panel on Climate Change (IPCC), and content published on the websites of Greenpeace International and Greenpeace Germany (Non-Governmental Organisations (NGOs)). Academic papers can be found either under a free open access (OA) policy, which does not require specific copyright permissions, or hidden behind a paywall, in which case the rules for content use are governed by the specific publisher. IPCC reports can be downloaded from the official website of IPCC<sup>6</sup> and used for personal, non-commercial purposes as long as the source is duly acknowledged. Translation of IPCC reports into German is managed by the German IPCC Coordination Office<sup>7</sup> and the translated content can be retrieved from their website. Content published on the two Greenpeace websites posed the most complex copyright case, mostly because of the

---

<sup>6</sup> <https://www.ipcc.ch/>

<sup>7</sup> <https://www.de-ipcc.de/index.php>

different copyright rules applicable to text on the one hand, and multimedia content on the other. Greenpeace has granted us approval to use images and videos that have been created by and are sole property of Greenpeace, as long as the content is used for research purposes [Gr22a, Gr22b] only.

### 3.2 Developing and implementing a metadata scheme

Metadata support corpus management and exploitation and constitute an integral part of linguistic research. They can be retrieved from the content description provided by the publisher, or obtained through data post-processing, including linguistic processing and information extraction. The metadata framework for the pilot corpus uses properties from the Dublin Core Metadata Initiative (DCMI Metadata Terms) as its backbone. We opted for the DCMI framework because it provides descriptive terms for data objects of different formats and constitutes a widely acknowledged standard that has been used in the description of both web and physical collections. This allows us to use the same schema for digitised collections which were not primarily designed to serve as web content.

At the time of the property selection, DCMI Metadata Terms entailed 55 properties [Du20], accompanied by a set of datatypes and vocabulary encoding schemes for the description of digital resources of various formats (including image, video, and audio). We selected 14 DCMI metadata terms: title, type, subject, publisher, contributor, identifier, rights, format, bibliographicCitation, rightsHolder, license, extent, created, accrualMethod. For a more detailed description of each term please see [Du20]. This information should be retrievable for each document in the corpus.

While the DCMI Metadata Terms provide a good selection of descriptive elements, they do not include fields for encoding all information of relevance to the project. Two containers of metadata properties were added to address this shortcoming: *linguisticInformation* and *mediaInformation*. The former is a container for project-relevant linguistic information gathered from both the given metadata, that is, metadata provided by the publisher, and for metadata derived by performing linguistic processing on the corpus. The latter gives information about the number and type of multimedia data objects embedded in a document. The two metadata containers are flexible and more properties can be added as necessary. At the moment, *linguisticInformation* stores information about genre, language, text type, status of content (archived or not, for more information see section 4.3), number of tokens, number of words, word types, content words, type-token ratio, lexical density, information about sentence, word, and token length, named entities and abbreviations. Each document was given a *filename* according to an agreed workable convention so that various media types can be linked to the document in which they are embedded. The intention is to apply this scheme to each document collected from the three data sources described in section 3.1. The collected metadata is added to each document and helps us build a profile of the whole corpus.

As already mentioned, some of the metadata properties are obtained by conducting linguistic processing and annotation of the content, which at the moment entails tokenization, part-of-speech (POS) tagging, dependency parsing, named entity recognition (NER), and abbreviation extraction.<sup>8</sup>

## 4 Data collection

This section elaborates on the data collection process from three sources: academic papers, the website of the International Panel on Climate Change (IPCC), and the content published on the websites of Greenpeace International and Greenpeace Germany.

### 4.1 Academic Papers

As a starting point for data collection for the pilot corpus, we used an article published by CarbonBrief titled “The most influential climate change papers of all time” [Pi15]. In this article, eight academic writings [AH97, Ca38, MW67, Ke76, No91, GZ00, HS06, HSR12] were highlighted as the most “cited” papers, which is a measure and an indication of how much impact the paper has in the scientific world. These seed papers ranged between the years 1896 to 2012, giving us a wide range of different climate change perspectives as the topic has evolved over time. We coined these eight papers as “seed papers” and these papers provided a way for us to extract information from them that would link us to other related academic works along the same topics across different years, providing a way for us to build a more comprehensible corpus.

**Building a corpus with the seed papers** We explored two methods for building a corpus using the eight seed papers that we have obtained: (1) checking the overlap of references between the seed papers, (2) extracted keywords and keyphrases from the academic papers were used as *seed terms* for search of more academic papers in the similar topic in Google Scholar.

With the first approach, we were not able to find any overlap between the references of the seed papers. Therefore, we did a search on Dimensions<sup>9</sup> for a list of the top citation references for each seed paper and from there we looked for overlapping citations. If a paper referenced to at least two seed papers, then that paper was taken to be included in the corpus. Based on this method, a total of 84 papers were initially collected.

<sup>8</sup> Any metadata obtained through content post-processing will be affected by the choice of tool used to perform this process. This paper demonstrates how such tools can be applied for metadata enrichment and does not focus on ways of improving their performance.

<sup>9</sup> <https://www.dimensions.ai/>

The second method was based on information extraction. The text content of the seed papers was extracted and analyzed with KeyBERT[Gr20]. KeyBERT provides integration of different pre-trained language models and since we only have academic papers in the English language, we opted for the model<sup>10</sup> developed initially by [RG19].

The top 10 keywords/keyphrases from each paper were extracted and these were grouped together according to semantic similarity. After the first iteration of extracting the keywords/keyphrases from the seed papers and grouping them together, a total of 9 clusters of keywords were formed. Each cluster of keywords was used as seed terms to search Google Scholar with *AND* operator between the terms and the top 20 results were taken and added to our collection. This iterative process was completed when we evaluated the corpus and found that we had obtained 1,812 academic papers using this method (see figure 1 for visualization of the process). The total number of academic papers downloaded was 1887, ranging from the years 1895 to 2022.

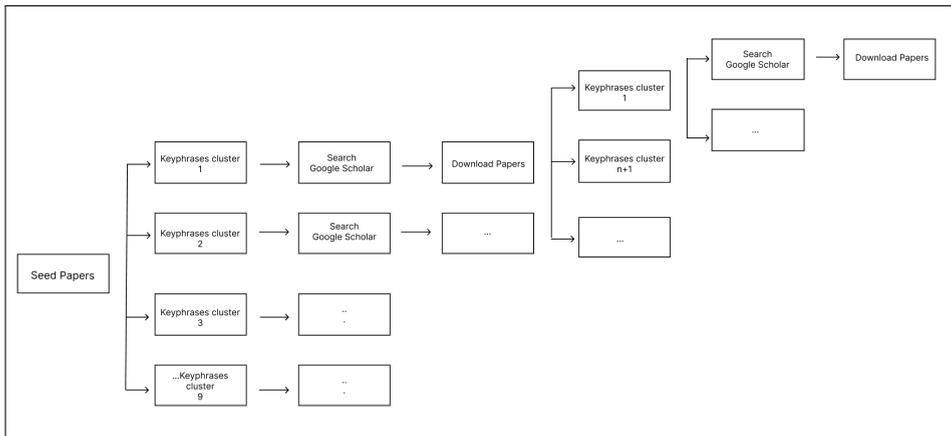


Fig. 1: An example of the iterative process for keywords extraction from seed papers to downloading the papers from Google Scholar.

## 4.2 Reports published by the International Panel on Climate Change (IPCC)

In the pilot corpus, we included the IPCC synthesis reports from each reporting period<sup>11 12</sup>. These reports are originally published in English; translations into German were collected

<sup>10</sup> all-MiniLM-L6-v2

<sup>11</sup> “Climate Change: The IPCC 1990 and 1992 Assessments”, “SAR Climate Change 1995: Synthesis Report”, “TAR Climate Change 2001: Synthesis Report”, “AR4 Climate Change 2007: Synthesis Report”, “AR5 Synthesis Report: Climate Change 2014”.

<sup>12</sup> At the time of writing this paper, the publication of the AR6 Synthesis Report is pending; only an outline of the report is available.

when available<sup>13</sup>. This means that the pilot corpus contains 5 synthesis reports in English and 3 synthesis reports or part of the synthesis report in German.

### 4.3 Greenpeace International and Greenpeace Germany

The web pages from Greenpeace International and Greenpeace Germany relevant to our project were retrieved by entering the prompt 'climate change' and 'Klimawandel' respectively in the search bar on each organisation's web site<sup>14</sup><sup>15</sup>. The search, performed in March 2022, returned 4057 links to web pages from Greenpeace International, of which 698 were hosted on the domain of Greenpeace International, while 3359 were archived and hosted on the domain of the Wayback Machine - Internet Archive<sup>16</sup>. We only include the 698 web pages hosted on the Greenpeace International domain in our pilot corpus in order to have a balanced number of tokens in each language (see table 1 for corpus size). From Greenpeace Germany, the search returned 1281 links to web pages.

## 5 Data parsing

The process described in section 4 resulted in files in two formats: PDF and HTML. This section discusses the tools used to parse the documents and extract relevant information.

### 5.1 Academic papers and IPCC reports

All academic papers and IPCC reports are saved and parsed as PDF files. For the scanned versions of PDF files, we use Tesseract [Ka07] to do OCR on the text and append the results as transparent text layers to the original PDF pages.

We combine the VILA [Sh22] and Resnet101 [He15] that was trained on DocBank [Li20] to parse PDF Files. VILA is a model for token sequence prediction, which does not predict the images in the document. Resnet101 takes as input the rendered image of each page of the document and identifies only the location of the figures on each page. The output label set is *Abstract, Author, Caption, Equation, Figure, Footer, List, Paragraph, Reference, Section, Table* and *Title*.

The models for parsing PDF files are run in an Online Learning [Ho18] framework and are loaded in Label Studio [Tk22] as a machine learning backend service. We import the

<sup>13</sup> At the moment, there are full translations of the synthesis reports for the years 2007 and 2014, and a translation of the Summary for Policymakers from 2001.

<sup>14</sup> <https://www.greenpeace.org/international/>

<sup>15</sup> <https://www.greenpeace.de>

<sup>16</sup> <https://archive.org/web/>

documents as a rectangular label object detection task into the front end, use the original models to make predictions for a small subset of documents, correct the prediction manually, and then fine-tune the models. Finally, we parse all documents using the updated models. The goal is not only to extract the text and other data objects from the PDF files, but also to retain the layout information of the documents so that we can examine the interactions between the data objects.

## 5.2 Greenpeace International and Greenpeace Germany

Since many of the webpages that we needed to download and parse were dynamic, we used Selenium<sup>17</sup> to retrieve the HTML from the collected links (see section 4.3) and BeautifulSoup [Ri07] to parse the content. When extracting the relevant data objects, we made sure to preserve the order of appearance of HTML elements containing important information, to extract their position on the web page, and to extract all HTML elements that might contain links to data objects in a modality different than text. This resulted in documents that mirror the output of the PDF parsing process described in section 5.1, hence allowing us to examine the interaction between various modalities.

## 6 Data Annotation

This section will highlight some of the methods used to annotate the parsed data. We will mainly discuss the annotation process for: (1) linguistic annotations including sentence splitting and tokenization, part-of-speech tagging and dependency parsing, (2) Named-entity recognition and (3) keywords/keyphrases extraction.

### 6.1 Linguistic annotation and Named-Entity Recognition (NER)

The linguistic annotation was done in a bottom-up approach: the text of a document was split into sentences, which were then run through an annotation pipeline. Three libraries constitute the annotation pipeline for English texts: spacy-stanza<sup>18</sup>, Stanford CoreNLP [Ma14]<sup>19</sup>, and SciSpacy[Ne19]<sup>20</sup>. Stanford CoreNLP is used to complement the named entity extraction for categories of named entities not covered by spacy-stanza. The extracted named entities from the English texts belong to the following 24 categories: PERSON, NORP, FAC, ORG, GPE, LOC, PRODUCT, EVENT, WORK-OF-ART, LAW, LANGUAGE, DATE, TIME, PERCENT, MONEY, QUANTITY, ORDINAL, CARDINAL, TITLE, CITY, IDEOLOGY,

<sup>17</sup> <https://github.com/SeleniumHQ/selenium>, v.3.14.0

<sup>18</sup> <https://spacy.io/universe/project/spacy-stanza>, running on stanza language model 1.4.1

<sup>19</sup> Version 4.4.0

<sup>20</sup> <https://github.com/allenai/scispacy>

RELIGION, CRIMINAL-CHARGE, and CAUSE-OF-DEATH. With SciSpacy we extract abbreviations from each sentence. The German documents were processed with stanza only, since both stanza and Stanford CoreNLP have only four categories of named entities for German language texts (ORG, PERSON, LOC, MISC).

In addition to saving the extracted annotations in a JSON file, each document with linguistic annotations is saved as a pickle file (German content) and both pickle file and spaCy object (English content). This step allows for consistency should we decide to extract linguistic patterns or another type of linguistic information.

The linguistic annotation served as the backbone of the linguistic information extracted and calculated for each document. The result of this process feeds back into the metadata, where the information is saved in the metadata container *linguisticInformation* as mentioned in section 3.2.

## 6.2 Keywords/Keyphrases extraction

As previously mentioned in section 4.1, KeyBERT was implemented to capture keywords and keyphrases that are semantically similar to the document content. The same approach was used to annotate the documents in our corpus. Since our corpus contains documents both in English and German, using pretrained language models in English is not enough. Therefore, a multilingual model <sup>21</sup> developed by [RG20] was used for German texts.

Through our implementation and experimentation of using KeyBERT for identifying seed terms as seen in section 4.1, we found that KeyBERT has the tendency to create noisy results, which did not have much effect on our results when using them as seed terms to search on Google Scholar, but would potentially have a much larger effect on keywords/keyphrases annotation of the data. Therefore, we combined the KeyBERT approach with the textrank approach proposed by [MT04]. We implemented textrank using PyTextRank [Na16] in the spaCy pipeline. Both English<sup>22</sup> and German<sup>23</sup> models were used through spaCy.

The keywords and keyphrases that had a semantic similarity score of 0.7 or higher were extracted using KeyBERT and the list was compared to the set that were extracted using PyTextRank; overlapping results were discarded and the final list of keywords and keyphrases was added to the metadata term *subject*.

## 7 Results

We present the results obtained from the data curation process of the pilot corpus and the type of multimedia data objects retrieved from each of the three data sources.

<sup>21</sup> paraphrase-multilingual-MiniLM-L12-v2

<sup>22</sup> en\_core\_web\_sm and en\_core\_web\_trf

<sup>23</sup> de\_core\_news\_sm and de\_dep\_news\_trf

**Academic papers** A total of 1,887 academic papers were collected and 15,461 images and figures were extracted from the PDFs. 1,095 equations and 2,207 tables were extracted and these are listed under “Other” in table 1. A total of over 43 million tokens were extracted from these documents.

**IPCC reports** The 5 English IPCC reports contained 315 images and figures and 104 tables/equations. A total of 496,477 tokens were extracted. For the 3 reports in German, 135 images and figures were extracted with 31 tables/equations and a total of 170,617 tokens were extracted.

**Greenpeace International and Greenpeace Germany** The 698 documents of Greenpeace International have 2066 embedded images, 123 embedded videos, 67 videos added to the content as hyperlinks, and 458 other types of multimedia objects. In 1, “Other” entails iframes, which are web pages embedded within another web page. In the context of the Greenpeace International corpus, iframes store videos, text, images, animations, dynamic charts, tweets, Facebook posts, Instagram posts, and files in a PDF format. The 1281 documents of Greenpeace Germany contained 2463 images and 14 videos. We retrieved 188 YouTube videos from Greenpeace International, whose total duration was 25.55 hours. The 14 videos of Greenpeace Germany amounted to 3.35 hours. Total number of tokens in the Greenpeace International transcripts were 157,115 and 27,586 tokens for Greenpeace Germany.

It is evident that of the total number of collected documents (3,874), the majority have embedded multimedia content (3,317), compared to text-only documents (557). This finding underpins the need for awareness of the various media types that support textual content, and for incorporating processing techniques that would enable researchers to analyse media content in the context of a document as a whole.

| Data Source              | Docs without MC* | Docs with MC* | Multimedia Content |        |       | # of Tokens |
|--------------------------|------------------|---------------|--------------------|--------|-------|-------------|
|                          |                  |               | Imgs/Figs          | Videos | Other |             |
| Academic Paper           | 100              | 1 787         | 15 461             | -      | 3 302 | 43 152 714  |
| IPCC reports EN          | 0                | 5             | 315                | -      | 104   | 496 477     |
| IPCC reports DE          | 0                | 3             | 135                | -      | 31    | 170 617     |
| Greenpeace International | 228              | 470           | 2 066              | 188    | 458   | 676 879     |
| Greenpeace Germany       | 229              | 1 052         | 2 463              | 14     | 463   | 645 962     |

\*MC: Multimedia Content

Tab. 1: Summary of the pilot corpus with number of extracted contents.

## 8 Discussion

This paper describes the process of building a multimodal pilot corpus comprising both a substantial number and a wide range of data types in documents. The pilot corpus was the starting point for developing methodologies that would allow us to better design and curate the ICCC. We believe such a multimodal dataset is needed as a starting point in order to gain further insights to the climate change topic by analyzing multimodal data and exploring the additional information that can be obtained.

**Possible use-case** One example use-case of such analysis can be the study of political views on a specific policy. With textual information, one can analyze the textual data with sentiment analysis to extract the sentiments regarding the policy. With the addition of multimodal data, we can also extract the sentiment in those data objects and determine if they play a role in strengthening the sentiment found in the textual data or not. Such insight can lead to a deeper understanding of the views and opinions concerning the specific policy.

**Lessons learned** We present some of the lessons learned in terms of data modelling of multimodal corpora, application of data annotation and information retrieval techniques, and challenges of working with a bilingual corpus.

It is evident that discarding multimedia data objects, as is common practice in corpus development, results in the possible loss of relevant information and eliminates the opportunity to investigate the interaction between data objects of different modalities. As seen in this paper, creating a data model that lends itself to modelling and analyzing interactions between different types of data objects presents another layer of complexity in the process of collecting, parsing, and analysing multimodal data.

Another important task was to enrich the corpus metadata by incorporating NLP tools for corpus annotation and information retrieval. While some of these techniques evidently enriched the metadata of the corpus, others performed well on one type of data, but not on another, highlighting the necessity of employing tools or models built for specific task in the specific target domain. More work will need to be done in this respect to seek out the appropriate tools and models and fine-tuning them to our domain-specific documents.

Lastly, we found that it is difficult to achieve entirely matching annotations for English and for German corpora, mostly because existing tools and models for processing German texts do not offer the same level of granularity and linguistic detail. Improving this situation is identified as a research desiderate in our future work.

As stated previously, the goal of this paper on the creation of ICCC pilot corpus is to explore the design and curation process of a multimodal corpus. The impact for future research is to provide a corpus-building methodology that will be applied in the next step of the research,

which is to expand the ICCG by looking further into collecting data from other sources in the climate change domain.

**Acknowledgments** We would like to thank Rasmus Beckmann and Jasper Korte of the Institute for Software Technology, German Aerospace Centre (DLR) for their assistance in identifying sources of data relevant to our research.

The research reported in this paper was conducted within the research project InsightsNet (<https://insightsnet.org/>) which is funded by the Federal Ministry of Education and Research (BMBF) under grant no. 01UG2130A.

## Bibliography

- [ADY11] Anne DiFrancesco, Darryn; Young, Nathan: Seeing climate change: The visual construction of global warming in Canadian national print media. *cultural geographies*, 18(4):517–536, 2011.
- [AH97] Arrhenius, S.; Holden, Edward S.: ON THE INFLUENCE OF CARBONIC ACID IN THE AIR UPON THE TEMPERATURE OF THE EARTH. *Publications of the Astronomical Society of the Pacific*, 9(54):14–24, 1897.
- [Ca38] Callendar, Guy Stewart: The artificial production of carbon dioxide and its influence on temperature. *Quarterly Journal of the Royal Meteorological Society*, 64(275):223–240, 1938.
- [Du20] DublinCore: DCMI Metadata Terms. 2020.
- [Gr20] Grootendorst, Maarten: , KeyBERT: Minimal keyword extraction with BERT., 2020.
- [Gr22a] Greenpeace, International: Email to Elena Volkanovska, 13 April. 2022.
- [Gr22b] Greenpeace, International: Email to Elena Volkanovska, 15 March. 2022.
- [GZ00] Guisan, Antoine; Zimmermann, Niklaus E: Predictive habitat distribution models in ecology. *Ecological modelling*, 135(2-3):147–186, 2000.
- [He15] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian: , Deep Residual Learning for Image Recognition, 2015.
- [Ho18] Hoi, Steven C. H.; Sahoo, Doyen; Lu, Jing; Zhao, Peilin: , Online Learning: A Comprehensive Survey, 2018.
- [HS06] Held, Isaac M; Soden, Brian J: Robust responses of the hydrological cycle to global warming. *Journal of climate*, 19(21):5686–5699, 2006.
- [HSR12] Hansen, James; Sato, Makiko; Ruedy, Reto: Perception of climate change. *Proceedings of the National Academy of Sciences*, 109(37):E2415–E2423, 2012.
- [Ka07] Kay, Anthony: Tesseract: An Open-Source Optical Character Recognition Engine. *Linux J.*, 2007(159):2, jul 2007.
- [Ke76] Keeling, Charles D; Bacastow, Robert B; Bainbridge, Arnold E; Ekdahl Jr, Carl A; Guenther, Peter R; Waterman, Lee S; Chin, John FS: Atmospheric carbon dioxide variations at Mauna Loa observatory, Hawaii. *Tellus*, 28(6):538–551, 1976.
- [LCJ20] Luo, Yiwei; Card, Dallas; Jurafsky, Dan: Detecting stance in media on global warming. *arXiv preprint arXiv:2010.15149*, 2020.
- [Li20] Li, Minghao; Xu, Yiheng; Cui, Lei; Huang, Shaohan; Wei, Furu; Li, Zhoujun; Zhou, Ming: , DocBank: A Benchmark Dataset for Document Layout Analysis, 2020.
- [Ma14] Manning, Christopher D; Surdeanu, Mihai; Bauer, John; Finkel, Jenny Rose; Bethard, Steven; McClosky, David: The Stanford CoreNLP natural language processing toolkit. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. pp. 55–60, 2014.

- [MM21] Mishra, Prakamy; Mittal, Rohan: NeuralNERE: Neural Named Entity Relationship Extraction for End-to-End Climate Change Knowledge Graph Construction. In: Tackling Climate Change with Machine Learning Workshop at ICML. 2021.
- [MT04] Mihalcea, Rada; Tarau, Paul: TextRank: Bringing order into text. In: Proceedings of the 2004 conference on empirical methods in natural language processing. pp. 404–411, 2004.
- [MW67] Manabe, SvUkURO; Wetherald, Richard T: Thermal equilibrium of the atmosphere with a given distribution of relative humidity. 1967.
- [Na16] Nathan, Paco: , PyTextRank, a Python implementation of TextRank for phrase extraction and summarization of text documents, 2016.
- [Ne19] Neumann, Mark; King, Daniel; Beltagy, Iz; Ammar, Waleed: ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. In: Proceedings of the 18th BioNLP Workshop and Shared Task. Association for Computational Linguistics, Florence, Italy, pp. 319–327, August 2019.
- [No91] Nordhaus, William D: To slow or not to slow: the economics of the greenhouse effect. *The economic journal*, 101(407):920–937, 1991.
- [Pi15] Pidcock, Roz: The most influential climate change papers of all time. 2015.
- [RG19] Reimers, Nils; Gurevych, Iryna: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2019.
- [RG20] Reimers, Nils; Gurevych, Iryna: Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2020.
- [Ri07] Richardson, Leonard: Beautiful soup documentation. Dosegljivo: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>. [Dostopano: 7. 7. 2018], 2007.
- [Sh22] Shen, Zejiang; Lo, Kyle; Wang, Lucy Lu; Kuehl, Bailey; Weld, Daniel S; Downey, Doug: VILA: Improving structured content extraction from scientific PDFs using visual layout groups. *Transactions of the Association for Computational Linguistics*, 10:376–392, 2022.
- [SP21] Stede, Manfred; Patz, Ronny: The climate change debate and natural language processing. In: Proceedings of the 1st Workshop on NLP for Positive Impact. pp. 8–18, 2021.
- [Tk22] Tkachenko, Maxim; Malyuk, Mikhail; Holmanyuk, Andrey; Liubimov, Nikolai: , Label Studio: Data labeling software, 2020–2022. Open source software available from <https://github.com/heartexlabs/label-studio>.
- [We16] Wessler, Hartmut; Wozniak, Antal; Hofer, Lutz; Lück, Julia: Global multimodal news frames on climate change: A comparison of five democracies around the world. *The International Journal of Press/Politics*, 21(4):423–445, 2016.

# Towards a User-Empowering Architecture for Trustability Analytics

Sebastian Bruchhaus,<sup>1</sup> Thoralf Reis,<sup>2</sup> Marco X. Bornschlegl,<sup>3</sup> Uta Störl,<sup>4</sup> Matthias Hemmje<sup>5</sup>

**Abstract:** Machine learning (ML) thrives on big data like huge data sets and streams from Internet of Things (IOT) devices. Those technologies are becoming increasingly commonplace in our day-to-day existence. Learning Autonomous Intelligent Actors (AIAs) impact our lives already in the form of, e.g. chat bots, medical expert systems, and facial recognition systems. Doubts concerning ethical, legal, and social implications of such AIAs consequently become increasingly compelling. Our society now finds itself confronted with decisive questions: Should we trust AI? Is it fair, transparent, and respecting privacy? An individual psychological threshold for cooperation with AIAs has been postulated. In Shaefer's words: "No trust, no use". On the other hand, ignorance of an AIA's weak points and idiosyncracies can lead to overreliance. This paper proposes a prototypical microservice architecture for trustability analytics. Its architecture shall introduce self-awareness concerning trustability into the AI2VIS4BigData reference model for big data analysis and visualization by borrowing the concept of a "looking-glass self" from psychology.

**Keywords:** Trust; Machine Learning; Digital Humanities; Foundation Model; Transparency; XAI

## 1 Introduction and Motivation

Individuals in our modern society are arguably compelled to accept the presence of Autonomous Intelligent Actors (AIAs) in their everyday environment. In literature AIAs are also referenced as "AI systems", "artificial agents", "autonomous systems", and sometimes "robots". Applied AI promises tremendous benefits, ranging from such diverse fields like healthcare to meteorology [Rei+22a; Rei+22b; Hig20]. This begs the question how society is going to integrate AIAs. A formal verification of their code is basically not unfeasible in most cases, because their ML models tend to be quite complex. GPT-3 was built in 2020 and has around 175 billion parameters [Bro+20]. On top of that, comprehension of its

---

<sup>1</sup> FernUniversität, DBIS, Universitätsstr. 1, 58097 Hagen, Germany, sebastian.bruchhaus@fernuni-hagen.de, <https://orcid.org/0000-0002-7783-2636>

<sup>2</sup> FernUniversität, MMIA, Universitätsstr. 1, 58097 Hagen, Germany, thoralf.reis@fernuni-hagen.de, <https://orcid.org/0000-0003-1100-2645>

<sup>3</sup> FernUniversität, MMIA, Universitätsstr. 1, 58097 Hagen, Germany, marco-xaver.bornschlegl@fernuni-hagen.de, <https://orcid.org/0000-0003-3789-5285>

<sup>4</sup> FernUniversität, DBIS, Universitätsstr. 1, 58097 Hagen, Germany, uta.stoerl@fernuni-hagen.de, <https://orcid.org/0000-0003-2771-142X>

<sup>5</sup> FernUniversität, MMIA, Universitätsstr. 1, 58097 Hagen, Germany, matthias.hemmje@fernuni-hagen.de, <https://orcid.org/0000-0001-8293-2802>

training data is key to understanding a model’s behavior. In practice such knowledge is often sketchy at best. These properties effectively turn all but the simplest AIAs into black boxes [Rud19]. The so-called “value alignment problem” results from this fundamental uncertainty [Had21]. Risk is a necessary prerequisite of trust [Jac+21]. Common sense forbids overreliance on automated decisions with the penalty of catastrophic consequences [Lif15]. Stakeholders resort to trust when they choose to accept the risks of an AIA although its benevolence or robustness cannot be proven rigorously. Trustworthy AIAs in data analytics and cyber-physical systems will arguably have an empowering effect on their human users [Rei+21]. They will be an important intermediate step towards real digital empathy between humans and AIAs [Bon+19].

## 1.1 Problem Statement and Research Questions

Trust is a necessity when working with AIAs but it is also a somewhat elusive concept. AI architects and engineers need standardized architectures and best practices that facilitate qualified trust. These architectures will serve as a foundation and yardstick for trustworthy AI software systems. This paper identifies the following challenges from the current scientific debate on trust and AIAs as research problems:

- (i) *Trust in AI has no canonical definition and lacks an universal vocabulary.*  
Terms like “transparent AI”, “explainable AI”, and even “responsible AI” or “ethical AI” may have subtly differentiated connotations with different authors. Researchers ought to refrain from using their own ad-hoc definitions. A common framework for trustability analytics needs to be established. There should be an unequivocal language and a sound understanding with rigorous models of trust as a solid foundation for future debate [Jac+21; Mil19; Lip18; Rud19].
- (ii) *Stakeholders must still rely on their intuition or educated guessing in order to determine the trustworthiness of an AI system.*  
The concept of trust has been described as “diffuse”, “disappointing”, and even “useless” by authors like O. Williamson [Wer18]. This paper intendeds to demonstrate that the latter verdict is an exaggeration. Trust clearly is an important asset for human society, which empowers us to cooperate and reach otherwise unachievable goals [Luh14]. There is, however, not yet a generally accepted metric for trustworthiness of AI. Trust is almost universally described as a highly individual and situational [KCW05]. Therefore it is a desirable feature for AIAs to address users individually and adjust to feedback.
- (iii) *There are no actionable guidelines on the practical engineering of trustworthy AIAs.*  
Despite of the many guidelines for ethical or trustworthy AI on the one hand, and a growing number of algorithms for ostensibly explainable, robust ML on the other, there is still hardly any practical, systematic advice on the implementation of trustworthy AIAs. Existing solutions seem somewhat insular. AIAs should be able to prove their

adherence to ethical and legal principles in the light of ongoing efforts to regulate ML for critical domains such as healthcare.

Foundation models are an excellent show case for these open problems, because of their immense practical usefulness despite a fundamental opacity [Bom+21]. Three questions shall be addressed in the following:

1. How can trustability of AIAs be practically modeled and analyzed?
2. How can a user empowering AIA maximize its trustability?
3. How can we design systems with regard to trustability?

The goal in answering those questions is to establish qualified trust or trustability in AIAs and an AIA architecture for big data analytics and visualization that proactively anticipates and maximizes its user's level of trust. Therefore it works out an actionable model of trust in section 3.1 after surveying the state of art of explainable AI (XAI) in section 2.3 and digital trust in section 3.2. It lays out a prototypical microservice architecture for trustability analytics in 3.2. This will introduce self-awareness concerning trustability into the AI2VIS4BigData reference model (sec. 2.2) for big data analysis and visualization in analogy to the "looking-glass self" theory from psychology described in 2.1.

## 2 State of the Art

A fair amount of literature is devoted to trust in AIAs. Yet there is still a shortage of practical research results such as complete and ready for use mathematical models. Stenton and Jensen come close to this with their blueprint model for trust in AI [SJ21]. Abbass lists a number of mathematical models of trust in [ALM16]: statistical models, Bayesian analysis, discrete models, belief models, fuzzy models, flow models, and optimization models. Some of which take the reputation of an AIA into consideration.

### 2.1 The Looking-Glass Self Theory

The psychologist Cooley developed a theory of an individual's self that he labeled "the looking-glass self". The eponymous "looking-glass" is an archaic term for a mirror. His theory is opposite the "self-verification theory" which states that we want others to see as we see ourselves. The self evolves by forming assumptions about the way that other individuals perceive us according to Cooley. His theory describes the human tendency to understand oneself through the judgements that others supposedly make. In short, humans form a mental model of their peers and how they perceive them. Then they adjust their self-view accordingly [McI07]. This happens in a three-step process [Sha04]:

1. Actors imagine how others must perceive them.
2. Actors consider how those others think of them.

3. Actors feel an affective reaction, e.g. pride or shame.

Consider an artist painting a self-portrait using a mirror, e.g. as depicted in the famous self-portraits of Johannes Gump and Norman Rockwell [Gos10; Roc60]. The artist (a stand-in for the AIA) uses a mirror (looking glass, the putative user’s perspective on the AIA) to paint an image of himself. A point of note here is that the affective reaction spurs a reaction and thus stimulates a feedback loop. This is sometimes known as the “Michelangelo phenomenon” in sociology. It causes individuals in romantic relationships to adjust their behavior in the direction to their assumed ideal self. In the context of AIAs, users’ mental processes in relation to the AIA must be emulated by a mathematical model of trust. Such a user models may be based on actual input data from the users or from first principles. The AIA can then adapt itself according to its supposed trustability score.

## 2.2 Big Data Analytics and AI2VIS4BigData

Training of ML models usually involves “big data”, i.e. data of high volume, variety, and velocity. Such a process tends to be lengthy and highly cost-intensive [Bom+21]. Many of the popular pre-trained foundation models, e.g. GPT-3, DALL-E 2, and BERT [Ram+22; Vas+17] do not enable end-users to inspect their training data. This can lead to underappreciation of algorithmic bias [Meh+21]. Applying the FAIR principles is arguably a step in the right direction [Jac+20], but this alone is insufficient to empower humans to a trustful collaboration with AIAs [Mon+20]. Data engineering for end-to-end machine learning usually happens in ML pipelines.

Reis developed a generic reference model called AI2VIS4BigData for such pipelines [Rei+22a]. It is an extension to the IVIS4BigData framework by Bornschlegl [BH21], using AI both directly and indirectly for analytics and user empowerment, e.g. by recommending analytics algorithms to subject matter experts without a background in data science [Rei+22b]. This abstract and versatile rendition of a generic data analytics workflow makes AI2VIS4BigData a suitable stand-in for a broad class of AIAs. Its focus on visual analysis for big data in combination with a strong connection to AI predisposes it for XAI use cases in particular.

AI2VIS4BigData describes a circular process repeating the following four steps of a big data analytics pipeline: Data Management & Curation, Analytics, Insight & Effectuation, and Interaction & Perception.

## 2.3 Trustability, Explanability, and Transparency of AI

Trust is primarily an interpersonal phenomenon. It has been studied extensively in psychology, sociology, and philosophy. Kracher gives an overview of the theories about trust in sociology with regard to computer science in [KCW05]. Trust is a prerequisite for human society

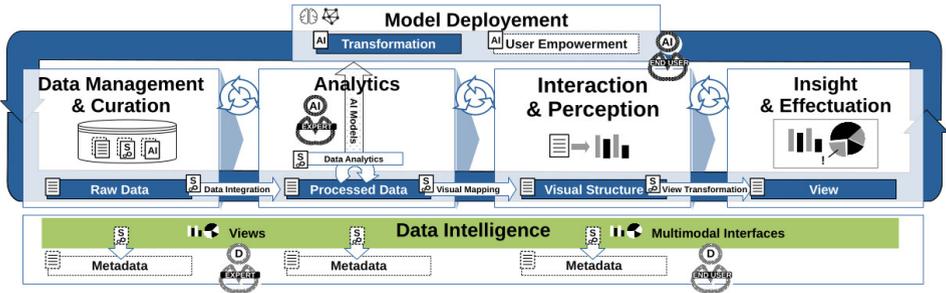


Fig. 1: Reis' AI2VIS4BigData reference model [Rei+22a]

in Niklas Luhman's opinion, as it reduces complexity for the individual [Luh14]. The sources of trust are twofold according to K. Wehrbach [Wer18]: "On one side is a belief rooted in some combination of rational and emotional factors; on the other is acceptance of uncontrolled risk." Three things are essential to trust: a trustor  $x$ , a trustee  $y$  which may be an inanimate object or AIA, and an element of uncertainty that introduces a risk. Its stakeholders must resort to trust. Hoff and Bashir describe a three-layered model of trust in automation that comprises dispositional, situational, and learned trust [HB14]. Better yet, users ought to be empowered by informed consent. Hence, they should depend on qualified trust that deals with risks transparently. Transparency has been identified as a fundamental prerequisite for trust [HBS11; Jac+21; Rud19; Bon+19]. This insight gave rise to the explainable AI (XAI) branch of AI research [GA19; Bar+20]. It is a truism in the field of explainable AI that understandability begets trust [HBS11; Mil19]. This paper follows the definitions in Barredo Arrieta's paper [Bar+20]: transparent systems are understandable by themselves while explainable systems present explanations as an accurate proxy to their users, etc. Yet the distinction between transparent and non-transparent ML models still lacks a satisfactory differentiation. The field of XAI is a very active research topic [GA19]. Several ostensibly transparent ML algorithms have been developed in recent years [DR20; BB21]. These are complemented by post-hoc explainers like SHAP and LIME for the analysis of black-box models [LL17; RSG16]. As all ML algorithms, these have individual strengths and weaknesses that can affect their trustability. These are hardly assessable – even by users with a background in ML. While it is reasonable that stakeholders are prone to put faith in systems they understand well, this idea should be taken with a grain of salt. There seems to be a trade-off with explanation completeness, as too complete explanations can even discourage users' trust in them [Kul+13; Pap+22]. Sceptics like Rudin and Lipton criticize current XAI methods and post-hoc explainers for black box models in particular [Lip18; Rud19].

### 3 Model Building

Abbass proposes a trust bus that has a Belief-Desire-Intention-Trust-Motivation (BDI-TM) architecture [ALM16]. The trust bus learns by passing messages between its six components. The constituent modules are these: actors and entities memory, trust production, identity management, intent management, emotion management, risk management, and complexity management. While Abbass' BDI-TM architecture is quite abstract and intended also for interactions between different AIAs, this paper will only consider it in the context of AI2VIS4BigData. Hence it focuses on AIA-user interactions as trustee and trustor. Therefore this paper adapts and simplifies the architecture significantly according to its use case, leading to a prototypical implementation.

#### 3.1 Trustability Analytics

The overall theme of this paper are trustability analytics for AI. Even if a satisfactory model for trust can be found, it is still not quite clear what the terms “trustability of AI” and by extension “trustability of AIA” exactly mean. In order to find an answer for research question 1, i.e. an actionable definition of trust, trustability analytics must be delineated. In the following, this paper will follow McCarthy's working definition [McC07]: “Intelligence is the computational part of the ability to achieve goals in the world.”, considering also the ISO's definition of risk as the “effect of uncertainty on an objective” in this context [ISO18]. Without a rigorous definition of AI the subsequent definition of its trustability, this paper must, however, remain preliminary.

AIAs are complex, emergent systems prone to unintended behavior [SY19]. The time-tested practice of software verification for critical settings guarantees that a program's execution aligns with its programmer's codified intentions. This is not yet feasible for AIAs. When the outcome of a computation involves uncertainty, the users knowingly or unknowingly accept some risk. That is inherently the case with self-modifying software such as any ML system during its training phase. A certain risk is consequently characteristic for AIAs.

Gerck offers different definitions of trust [Ger02]. The simplest form is: “Trust is to rely upon actions at a distance.” This describes the concept of trust as reliance. Another is “qualified reliance on information”. Gerck's definitions connect information theory and – by extension probability theory – with risk and its role as a precondition for trust. These thoughts on trust are rather insightful but too abstract to be directly applicable for software implementation. In summary, uncertainty begets risk and by extension (dis-)trust. Transparency on the other hand reduces uncertainty, or rather epistemic uncertainty, for the user (see also fig. 2).

If trust is “qualified reliance”, how is it to be qualified conveniently? In line with the research question 1 Stanton and Jensen give an answer to the question: “How does our evolutionarily ingrained and socially conditioned trust mechanism respond to machines?” They present a mathematical model for AI trustability  $T(u, s, a)$  depending on a specific user  $u$ , an AI

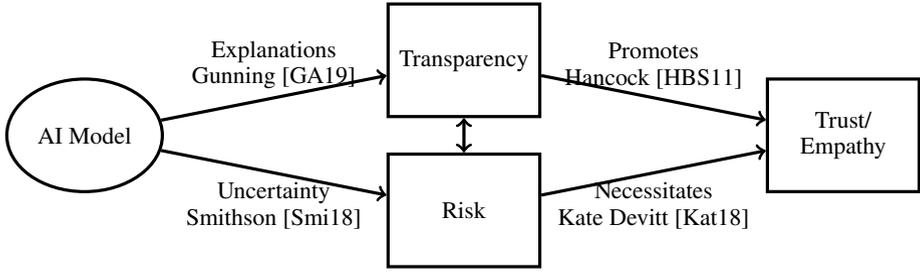


Fig. 2: Risk and transparency as influence factors for trust

system  $s$ , and a specific context  $a$  [SJ21]. They define the User Trust Potential  $UTP(u)$ . This subsumes the cultural, individual and otherwise subjective attributes of a user  $u$ . An AI system designer has little influence over  $UTP$  beyond precluding certain potential users from using his system. Of course, predominantly positive experiences with AI will have a positive impact on society-wide  $UTP$  in the long run.

There is also a less arbitrary and subjective factor of trustability that is significantly determined by the design of software architecture. This factor is Perceived System Trust Potential  $PST(u, s, a)$  in a system  $s$  within a context  $a$ . This has an extensive overlap with learned and situational trust in Hoff's and Bashir's model. There is arguably also an additional objective component of qualified trust depending on  $s$  and  $a$ . The overall likelihood of trust is defined as:

$$T(u, s, a) = UTP(u) \cdot PST(u, s, a). \quad (1)$$

$PST$  itself is defined as an arbitrary function  $g$  of user experience  $UX$  and perceived technical trustworthiness  $PTT$ : The latter is the sum of nine system characteristics  $ptt_c$ : accuracy, reliability & resiliency, objectivity & security, explainability, safety & accountability, and privacy. Thus

$$PST = g(UX, PTT), \quad PTT = \sum_{c=1}^9 ptt_c. \quad (2)$$

Each characteristic is the product of its pertinence  $p_c$  and sufficiency  $s_c$ :

$$ptt_c = p_c \cdot s_c. \quad (3)$$

Each of the nine characteristics is assigned a relative Pertinence

$$p_c = \frac{q_c}{\sum_{k=1}^9 q_k}. \quad (4)$$

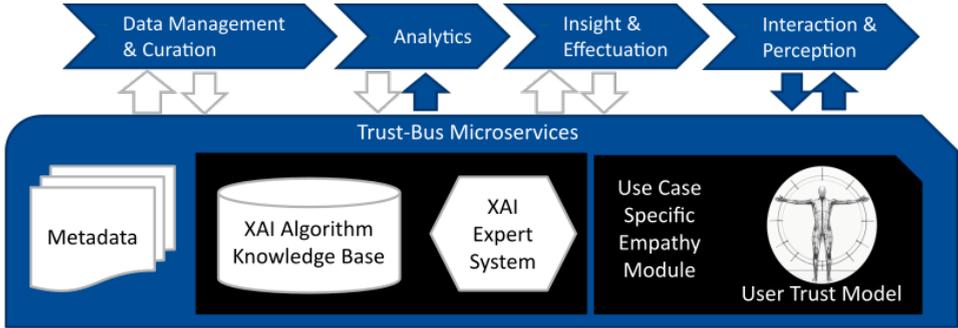
for an absolute pertinence  $q_c$  and a sufficiency  $s_c$  for a trustworthiness metric  $m_c$  and a perceived risk  $r_a$  in the given context:

$$s_c = m_c / r_a. \quad (5)$$

Obviously, Stanton’s and Jensen’s model of trust in AI is far from complete.  $UTP$ ,  $g$ ,  $q_c$ ,  $m_c$ , and  $r_a$  remain uncertain. They address this by posing a number of research questions. Most are concerned with deducing parameter values from first principles. As this is not yet possible, a prototypical implementation must resort to preliminary measures like user and expert surveys or estimation in lieu of deduction. This should happen on a case by case basis. This paper proposes an XAI knowledge base (KB) that compiles these values for different ML algorithms and will be updated regularly.

The parameters must be deduced from first principles whenever possible. But what are good candidates for those principles? Subjective trust is not enough to build robust and beneficial AIAs. Attributes like reproducibility, transparency, and fairness must be monitored and quantified, if objective facts are to guide the decision to trust an AIA. Users must be made aware of risks that they have to accept [HBS11; Kat18; Smi18]. AIAs not only need to demonstrate the “what?” to their stakeholders but must also be able to explain the “how?” of their decisions, in order to achieve transparency. The answer to “how?” will come from systematically captured and analyzed meta-data presented in unison with the model’s main results. Concerning the quality of training data, AIAs’ purpose is to receive complex tasks from a human user. They represent data in action [ASR18] while FAIR is more concerned with data reusability for scientific knowledge mining and synthesis. Other metrics, e.g. for fairness or robustness, can be chosen according to the individual use case.

Fig. 3: Trust bus architecture for AI2VIS4BigData, adds a feedback loop that facilitates self-assessment through empathy with its users (Parts of this image were created with the assistance of DALL-E 2.)



### 3.2 Trust Bus

The research question 2 asks for the implementation of a software service for trustability analytics. This service cannot be a data sink, i.e. the meta-data must be processed somehow. This paper suggests a trust bus architecture to synthesize qualified trust in the trustor. The trust bus architecture was introduced in section 3.1. See figure 3 for its structure. The authors of this paper envision a rule based expert system (ES) for this task. This system will be akin to that in [Rei+22b]. This ES can query an analogue to the looking-glass self for AIAs.

Thus, the service can self-assess its trustworthiness in the eyes of its users. It computes the trust value  $T(u, s, a)$  from eq. 1. The goal is to empower users by suggesting trustworthy, transparent, and explainable algorithms according to their individual needs. Therefore, the ES needs knowledge about the specific user  $u$  and the situation  $a$  that it addresses.

| BDI-TM Module              | Function  |
|----------------------------|---|
| Actors and Entities Memory | stores information about past interactions between trustor and trustees for later evaluation by the trust production module |
| Trust Production           | mechanisms to gauge trustworthiness of the AIA  |
| Identity Management        | identifies users or groups of users for appropriate handling by emotion and intent management                               |
| Intent Management          | estimates users' intent, predicts possible consequences of trust and informs the trust production module accordingly        |
| Emotion Management         | models and learns affective states of the user from interaction   |
| Risk Management            | manages a task-specific risk registry and analyzes uncertainties in relation to possible risks                              |
| Complexity Management      | estimates task complexities and the users' ability to make reasonable decisions under these circumstances                   |

Tab. 1: Components of the trust bus BDI-TM architecture from [ALM16]

ES takes care of the *trust production* in table 1. It requests data from the knowledge base KB that stores information about explainability and the specific risks encodes as parameters of ML algorithms. The task specific empathy module holds information about *complexity management* that it passes on to ES and evaluates whether a given task has special requirements concerning fairness, transparency, etc. The metadata store (MS) holds the *actors and entities memory*, but also deals with *identity management*.

The trust bus is intended to be a reference architecture for trustability analytics and hence ought to be fairly universal. Therefore, it needs to interact with all steps of the AI2VIS4BigData analytics pipeline in figure 1. It shall recommend analytics, pre-processing and visualization methods in order to maximize the users' trust  $T(u, s, a)$  from equation 1, which it provides for different users and situations by means of modularization. The system gauges its own value for  $T(u, s, a)$  with a use case specific "empathy module" (EM). This module represents the mirror or looking-glass in the looking-glass self analogy. It has a user trust model that estimates  $UTP(u)$ . It also predicts the values for  $PST(u, s, a)$  to the ES. The EM's input data come from ES, MS, and KM. Cues from its end users are taken in the last step of AI2VIS4BigData: "insight and effectuation". This feedback is taken into account along with the other metadata when updating this trust model in a feedback loop. It is possible that the system asks the user to intervene when it passes a predefined uncertainty level.

### 3.3 Towards a Prototypical Implementation

Steps towards a prototypical implementation of the reference architecture are underway. An experimental system based on AI2VIS4BigData for the explanation of sentiment analysis of natural language texts was implemented. This system uses the visual presentation of feature importance calculated by the post-hoc explanation algorithms LIME and SHAP on variants of the transformer model BERT [Vas+17]. However, it turns out to be highly sensitive to certain data errors like word permutation and duplication, sometimes resulting in incomprehensible explanations. A later and structurally similar system is used for the recognition of emergent, formerly unknown named entities (NER) in medical texts. It forgoes explicit explanations but adds training on adversarial examples to its ML-pipeline for improved robustness. More use-cases involving foundation models for semi-autonomous visual information extraction of highly non-uniform documents and the generation of ontologies from biomedical research documents are being developed. A commonality between these systems is their emphasis on interaction with a human in the loop for result quality control. These systems and more provide input for the knowledge base KB on different approaches to XAI and test beds for the trust bus. As a next step and future research, a data schema for the knowledge base KB must be modeled. Concurrently an expert system will be implemented using miniKanren [Wil20; FBK05]. Together, these will build a first iteration of a prototypical trust bus as outlined above. The rules for this expert system ES will be evaluated in cooperation with users and domain experts.

## 4 Summary and Outlook

This paper has posed a number of open research questions concerning technical solutions of trust issues arising from the emergence of AIAs in big data analytics. It seeks to answer these questions by the nascent field of trustability analytics which is multidisciplinary and is located in digital humanities. The authors proposed a software service that takes cues from the theory of a looking-glass self. Such a service may fill the role of an architectural blueprint for future designs of trustworthy AIAs in big data analytics.

Many details of such a trustability service remain for future research because of the preliminary condition of the scientific debate. In particular, parameters cannot yet be derived from first principles. First and foremost, a knowledge base for XAI algorithms with parameters for Stanton's and Jensen's trust model has to be compiled before the trust bus architecture for AI2VIS4BigData can be implemented.

“Trustability analytics” stand for a reductionist concept of trust that can be reasoned about rigorously and which leads towards practical software engineering. This paper's outlook on digital trust will undoubtedly leave something to be desired for humanities' scholars, but as George Box famously quipped: “All models are wrong, but some are useful.”

The authors of this paper hope that the proposed architecture will be a step towards practical application and a software prototype in the near future.

## References

- [ALM16] Hussein A. Abbass, George Leu, and Kathryn Merrick. “A Review of Theoretical and Practical Challenges of Trusted Autonomy in Big Data”. In: *IEEE Access* 4 (2016), pp. 2808–2830. DOI: 10.1109/access.2016.2571058.
- [ASR18] Hussein A. Abbass, Jason Scholz, and Darryn J. Reid. “Foundations of Trusted Autonomy: An Introduction”. In: *Foundations of Trusted Autonomy*. Springer International Publishing, 2018, pp. 1–12. DOI: 10.1007/978-3-319-64816-3\_1.
- [Bar+20] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information Fusion* 58 (2020), pp. 82–115. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2019.12.012.
- [BB21] Przemyslaw Biecek and Tomasz Burzykowski. *Explanatory Model Analysis. Explore, Explain, and Examine Predictive Models*. CRC PRESS, 2021. ISBN: 9780367135591.
- [Bom+21] Rishi Bommasani et al. *On the Opportunities and Risks of Foundation Models*. 2021.
- [Bon+19] Raymond Bond et al. “Digital empathy secures Frankenstein’s monster”. In: *Proceedings of the 5th Collaborative European Research Conference (CERC 2019)*. Vol. 2348. Apr. 2019, pp. 335–349.
- [BH21] Marco Xaver Bornschlegl and Matthias L. Hemmje. “Supporting Data Science in Automotive and Robotics Applications with Advanced Visual Big Data Analytics”. In: *Advances in Data Science: Methodologies and Applications*. Ed. by Gloria Phillips-Wren, Anna Esposito, and Lakhmi C. Jain. Cham: Springer International Publishing, 2021, pp. 209–249. ISBN: 978-3-030-51870-7. DOI: 10.1007/978-3-030-51870-7\_11.
- [Bro+20] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. DOI: 10.48550/ARXIV.2005.14165.
- [DR20] Arun Das and Paul Rad. “Opportunities and Challenges in Explainable Artificial Intelligence (XAI): a Survey”. In: *CoRR* (2020).
- [FBK05] Daniel P. Friedman, William E. Byrd, and Oleg Kiselyov. *The Reasoned Schemer*. The MIT Press, 2005. ISBN: 0262562146.
- [Ger02] Ed Gerck. “Trust as Qualified Reliance on Information, Part I”. en. In: (2002). DOI: 10.13140/RG.2.2.22646.04165.

- [Gos10] Helena Goscilo. “The Mirror in Art: Vanitas, Veritas, and Vision”. In: *Studies in 20th & 21st Century Literature* 34.2 (June 2010). DOI: 10.4148/2334-4415.1733.
- [GA19] David Gunning and David Aha. “DARPA’s Explainable Artificial Intelligence (XAI) Program”. In: *AI Magazine* 40.2 (June 2019), pp. 44–58. DOI: 10.1609/aimag.v40i2.2850.
- [Had21] Dylan Hadfield-Menell. “The Principal-Agent Alignment Problem in Artificial Intelligence”. PhD thesis. EECS Department, University of California, Berkeley, Aug. 2021.
- [HBS11] P. A. Hancock, D. R. Billings, and K. E. Schaefer. “Can You Trust Your Robot?” In: *Ergonomics in Design: The Quarterly of Human Factors Applications* 19.3 (July 2011), pp. 24–29. ISSN: 1064-8046. DOI: 10.1177/1064804611415045.
- [Hig20] High-Level Expert Group on AI. *White Paper on Artificial Intelligence. a European approach to excellence and trust*. eng. Report. Brussels: EU Kommission, Feb. 2020.
- [HB14] Kevin Anthony Hoff and Masooda Bashir. “Trust in Automation”. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 57.3 (Sept. 2014), pp. 407–434. DOI: 10.1177/0018720814547570.
- [ISO18] ISO Central Secretary. *Risk management – Guidelines*. en. Standard ISO 31000:2018. Geneva, CH: International Organization for Standardization, 2018.
- [Jac+20] Annika Jacobsen et al. “FAIR Principles: Interpretations and Implementation Considerations”. In: *Data Intelligence* 2.1-2 (Jan. 2020), pp. 10–29. ISSN: 2641-435X. DOI: 10.1162/dint\_r\_00024.
- [Jac+21] Alon Jacovi et al. “Formalizing Trust in Artificial Intelligence: Prerequisites, Causes and Goals of Human Trust in AI”. In: FAccT ’21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 624–635. ISBN: 9781450383097. DOI: 10.1145/3442188.3445923.
- [Kat18] S. Kate Devitt. “Trustworthiness of Autonomous Systems”. In: *Studies in Systems, Decision and Control* (2018), pp. 161–184. ISSN: 2198-4190. DOI: 10.1007/978-3-319-64816-3\_9.
- [KCW05] Beverly Kracher, Cynthia Corritore, and Susan Wiedenbeck. “A foundation for understanding online trust in electronic commerce”. In: *Journal of Information, Communication and Ethics in Society* 3 (Aug. 2005), pp. 131–141. DOI: 10.1108/14779960580000267.
- [Kul+13] T. Kulesza et al. “Too much, too little, or just right? Ways explanations impact end users’ mental models”. In: *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* (2013), pp. 3–10. DOI: 10.1109/VLHCC.2013.6645235.

- [Lif15] Future of Life Institute. *An Open Letter on AI*. <https://futureoflife.org/ai-open-letter/>. Accessed: 2020-8-25. 2015.
- [Lip18] Zachary C. Lipton. “The mythos of model interpretability”. In: *Communications of the ACM* 61.10 (Sept. 2018), pp. 36–43. DOI: 10.1145/3233231.
- [Luh14] Niklas Luhmann. *Vertrauen – Ein Mechanismus der Reduktion sozialer Komplexität*. 5th ed. utb GmbH, Feb. 2014. DOI: 10.36198/9783838540047.
- [LL17] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774.
- [McC07] John McCarthy. *WHAT IS ARTIFICIAL INTELLIGENCE?* 2007. URL: <http://www-formal.stanford.edu/jmc/whatisai.html> (visited on 04/01/2021).
- [McI07] L. McIntyre. *The Practical Skeptic: Core Concepts in Sociology*. McGraw-Hill Companies, Incorporated, 2007. ISBN: 978-0-07-340415-8.
- [Meh+21] Ninareh Mehrabi et al. “A Survey on Bias and Fairness in Machine Learning”. In: *ACM Comput. Surv.* 54.6 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3457607.
- [Mil19] Tim Miller. “Explanation in artificial intelligence: Insights from the social sciences”. In: *Artificial Intelligence* 267 (Feb. 2019), pp. 1–38. DOI: 10.1016/j.artint.2018.07.007.
- [Mon+20] Barend Mons et al. “The FAIR Principles: First Generation Implementation Choices and Challenges”. In: *Data Intelligence* 2.1-2 (Jan. 2020), pp. 1–9. ISSN: 2641-435X. DOI: 10.1162/dint\_e.00023.
- [Pap+22] Andrea Papenmeier et al. “It’s Complicated: The Relationship between User Trust, Model Accuracy and Explanations in AI”. In: *ACM Trans. Comput.-Hum. Interact.* 29.4 (Mar. 2022). ISSN: 1073-0516. DOI: 10.1145/3495013.
- [Ram+22] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. DOI: 10.48550/ARXIV.2204.06125.
- [Rei+21] Thoralf Reis et al. “Towards Modeling AI-based User Empowerment for Visual Big Data Analysis”. In: *BIRDS+WEPIR @ CHIIR 2021*. Ed. by Ingo Frommholz et al. Vol. 2863. CEUR Workshop Proceedings. CEUR-WS.org, Mar. 2021, pp. 67–75.
- [Rei+22a] Thoralf Reis et al. “A Service-based Information System for AI-supported Health Informatics”. In: *2022 IEEE 5th International Conference on Big Data and Artificial Intelligence (BDAl)*. 2022, pp. 99–104. DOI: 10.1109/BDAl56143.2022.9862611.
- [Rei+22b] Thoralf Reis et al. “Supporting Meteorologists in Data Analysis through Knowledge-Based Recommendations”. In: *Big Data and Cognitive Computing* 6.4 (Sept. 2022), p. 103. DOI: 10.3390/bdcc6040103.

- [RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 2016, pp. 1135–1144.
- [Roc60] Norman Rockwell. “Triple self-portrait”. In: *The Saturday Evening Post* (Feb. 1960).
- [Rud19] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature Machine Intelligence* 1.5 (May 2019), pp. 206–215. DOI: 10.1038/s42256-019-0048-x.
- [SY19] P.J. Scott and R.V. Yampolskiy. “Classification Schemas for Artificial Intelligence Failures”. In: *Delphi - Interdisciplinary Review of Emerging Technologies* 2.4 (2019), pp. 186–199. DOI: 10.21552/delphi/2019/4/8.
- [Sha04] Leigh S. Shaffer. “From mirror self-recognition to the looking-glass self: Exploring the Justification Hypothesis”. In: *Journal of Clinical Psychology* 61.1 (2004), pp. 47–65. DOI: 10.1002/jclp.20090.
- [Smi18] Michael Smithson. “Trusted Autonomy Under Uncertainty”. In: *Studies in Systems, Decision and Control* (2018), pp. 185–201. ISSN: 2198-4190. DOI: 10.1007/978-3-319-64816-3\_10.
- [SJ21] Brian Stanton and Theodore Jensen. *Trust and Artificial Intelligence*. en. Mar. 2021. URL: [https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=931087](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=931087) (visited on 01/09/2023).
- [Vas+17] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.
- [Wer18] Kevin Werbach. *The blockchain and the new architecture of trust*. Information Policy. London, England: MIT Press, Nov. 2018.
- [Wil20] Brandon T. Willard. *miniKanren as a Tool for Symbolic Computation in Python*. 2020. DOI: 10.48550/ARXIV.2005.11644.

Workshop on Data Engineering for Data Science  
(DE4DS)



# “FAIR” is not enough – A Metrics Framework to ensure Data Quality through Data Preparation

Valerie Restat,<sup>1</sup> Meike Klettke,<sup>2</sup> Uta Störl<sup>3</sup>

**Abstract:** Data-driven systems and machine learning-based decisions are becoming increasingly important and are having an impact on our everyday lives. The prerequisite for good results and decisions is good data quality, which must be ensured by preprocessing the data. For domain experts, however, the following difficulties arise: On the one hand, they have to choose from a multitude of different tools and algorithms. On the other hand, there is no uniform evaluation method for data quality. For this reason, we present the design of a framework of metrics that allows for a flexible evaluation of data quality and data preparation results.

**Keywords:** data quality; metrics; evaluation; data preparation

## 1 Introduction

Real data is rarely error-free. To use it for analysis and to ensure the quality of the derived decisions, data preparation is necessary [Ab16]. A variety of different tools exist for this purpose, which are often combined in a preprocessing pipeline. The difficulty, however, is to choose from this range of tools and possible combinations depending on the data and the use case. More support is required for domain experts without in-depth IT knowledge. As a first step in this direction, we see the need for a framework of metrics to assess data quality. Such a framework would generate a number of advantages:

- The results of different data preparation tools become comparable.
- New solutions can be evaluated.
- Data quality is measurable and thus the quality of analyses and automated decisions can be ensured.

To the best of our knowledge, such a systematic framework does not yet exist. Hence, in this paper we propose a framework of metrics that takes into account many different aspects of data quality. Depending on the use case, this allows metrics to be selected that are suitable for the scenario in question.

After the summarization of related work in Section 2, we present this framework in Section 3. It consists of two dimensions, which are described in Section 3.1 and Section 3.2

---

<sup>1</sup> University of Hagen, Universitätsstr. 1, 58097 Hagen, Germany [valerie.restat@fernuni-hagen.de](mailto:valerie.restat@fernuni-hagen.de)

<sup>2</sup> University of Regensburg, Bajuwarenstr. 4, 93053 Regensburg, Germany [meike.klettke@ur.de](mailto:meike.klettke@ur.de)

<sup>3</sup> University of Hagen, Universitätsstr. 1, 58097 Hagen, Germany [uta.stoerl@fernuni-hagen.de](mailto:uta.stoerl@fernuni-hagen.de)

subsequently. The measurement of the proposed metrics is defined in Section 3.3. We conclude the paper in Section 4 and present future work.

## 2 Related Work

Well-known principles are the FAIR principles (*Findable, Accessible, Interoperable* and *Re-usable*) for automatically finding, using, and re-using data [Wi16]. However, these do not take into account the quality of the data, which must be ensured by preprocessing.

In software engineering, there is a standard (ISO/IEC 9126) that defines software quality criteria, and various metrics have been proposed to prove each criterion. For data, there is no such standardization yet. There is some initial research work here.

A variety of different definitions exist for data quality, including timeliness, currency, accuracy and completeness, credibility, and presentation quality [CZ15; Si12]. In addition, aspects such as relevance and fairness must be considered [CZ15; Pi20; SS20]. We have provided a more detailed description of the different dimensions of data quality in [RKS22]. Since the focus of this paper is on the metrics framework, only the most important aspects are mentioned here.

The multitude of different definitions also leads to the fact that there is no standardized evaluation. Different metrics exist (e.g. [HH16] and [BM11]), but these only cover a few aspects of data quality. To the best of our knowledge, a systematic framework that takes into consideration many different data quality dimensions does not yet exist.

Data quality validation is also addressed in Schelter et al. [Sc18]. The authors describe a system for automating the verification of data quality. The following aspects are considered: Completeness, consistency, and accuracy. The system scales well for large data sets and provides users with a declarative API. By combining common quality constraints and custom validation code, it offers many possibilities for evaluation. However, our classification is even more comprehensive. It provides a more detailed classification of the metrics and distinguishes more precisely to what extent domain knowledge is required.

In Polyzotis et al. [Po17], challenges related to machine learning are addressed, with particular reference to the deviation between serving and training data. Our framework can be used to compare the quality of serving and training data. In future work, we also want to explore how we can further extend the framework to address the challenges that specifically arise in context of machine learning.

## 3 Metrics Framework

Our proposed framework consists of two dimensions, which are explained in more detail in the following subsections:

- Horizontal dimension – The status of data preparation (Section 3.1)
- Vertical dimension – The extent to which ground truth is known (Section 3.2)

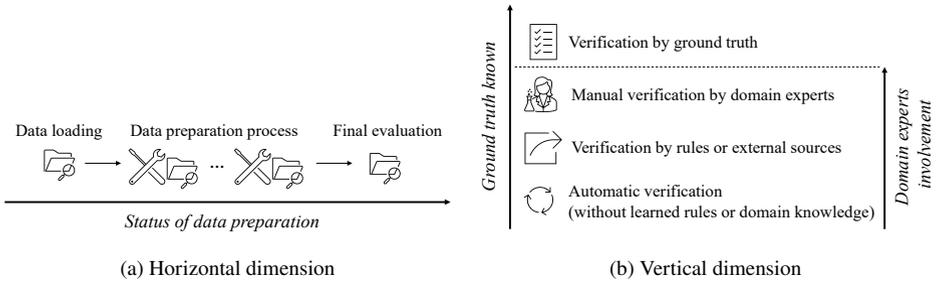


Fig. 1: Metrics framework

The horizontal dimension, shown in Figure 1a, describes the status of data preparation. Some aspects have to be considered already at the time of data collection or loading. Other aspects are continuously improved by preprocessing the data. At the end of preprocessing, a final evaluation should be performed again.

On the vertical dimension, shown in Figure 1b, a distinction is made between the extent to which ground truth is known. It is only rarely the case that ground truth is completely available. In other cases, the evaluation depends on the degree to which domain experts are involved. If these are available, a manual evaluation can be performed. Otherwise, the data quality must be checked using rules or external sources. If these are also not available, an automated check can be performed in individual cases.

In the following, the two dimensions and the corresponding metrics are explained in more detail. It is described which aspects of data quality must be taken into account. Subsequently, Section 3.3 outlines how the metrics can be measured.

### 3.1 Horizontal Dimension

First, the horizontal dimension is considered. It defines the point in time when the evaluation of data quality takes place.

**Data loading** At the beginning of data preparation, the data must be loaded. In the process, some aspects of data quality must first be checked. This includes the following aspects, which have been mentioned in Section 2:

- Timeliness
- Credibility
- Relevance

These aspects must be checked at the beginning and are prerequisites for data quality. If they are not met, an improvement by e.g. data cleaning is not possible. For example, if the data is not suitable for the application purpose, data preparation will not improve it either. New data must then be collected instead.

**Data preparation process** Other aspects of data quality are only achieved through the preparation process itself. Continuous evaluation is possible here. After each step, it can be checked whether the data quality has improved. At the end of data preparation, a final evaluation should be performed. In [Re22], we have already declared an extensive error classification that covers a variety of different data quality aspects. Therefore, we base these kinds of metrics on the error classification presented in [Re22], illustrated in Figure 2. The different types of errors are shown in Table 1. In the first step, we focus on single relation levels to assess the quality of individual data sets. In future work this classification should be extended.

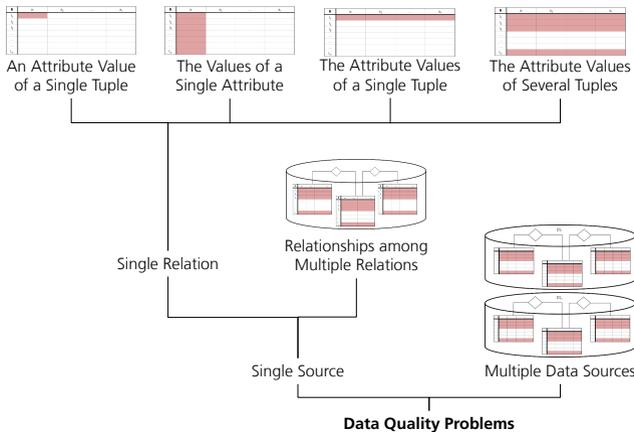


Fig. 2: Error classification specified in [Re22]

**Final evaluation** At the end of the data preparation process, all aspects from Table 1 should be checked again. As an additional point, depending on the use case, the evaluation of the presentation quality, mentioned in [RKS22], is also conceivable here.

In the following section, the metrics are described and subdivided in the vertical dimension.

### 3.2 Vertical Dimension

This dimension describes the extent to which ground truth is incorporated into the metrics. It also indicates how much domain experts are involved in the verification.

Tab. 1: Error types specified in [Re22]

| Level                                  | Error Type                                |
|--|---|
| An Attribute Value of a Single Tuple   | Missing value                             |
|  | Syntax violation                          |
|  | Interval violation                        |
|  | Set violation                             |
|  | Misspelled error                          |
|  | Inadequate value to the attribute context |
|  | Value items beyond the attribute context  |
|  | Meaningless Value                         |
|  | Erroneous entry                           |
| The Values of a Single Attribute       | Uniqueness value violation                |
|  | Synonyms existence                        |
|  | Outlier                                   |
|  | Missing Attribute                         |
| The Attribute Values of a Single Tuple | Semi-empty tuple                          |
|  | Inconsistency among attribute values      |
|  | Irrelevant observation                    |
| The Attribute Values of Several Tuples | Redundancy about an entity                |
|  | Inconsistency about an entity             |
|  | Bias                                      |
|  | Noise                                     |

**Verification by ground truth** When ground truth is present, automated matching can take place. However, it is only rarely the case that ground truth is fully available. The creation may also require a high level of involvement of domain experts. In addition, it must be noted that there is no ground truth for some quality aspects. Along the horizontal dimension, this concerns mainly the aspects of data loading, as will be shown in Section 3.2.1. In the final evaluation there is also one aspect for which no ground truth can be determined, shown in Section 3.2.3.

**Manual verification by domain experts** If ground truth is not available, human involvement is necessary. Normally, domain experts know the data best and are therefore in the

best position to judge the quality of the data. However, this process is very time consuming and therefore mostly not feasible.

**Verification by rules or external sources** For this reason, the next step is to permit domain experts to specify rules that can be used to check errors in the data. In some cases, it is also possible to use external sources. It should be noted that there are a number of tools that use pattern enforcement or machine learning models to detect errors. HoloDetect [He19] and Raha [Ma19] are examples for such tools. Theoretically, those models or rules learned by models can also be used for evaluation. The result of these could then be compared with the results present. However, it is difficult to state about which results are more correct. For this reason, it was not considered in the present framework.

**Automatic verification (without learned rules or domain knowledge)** If no rules or external sources are available, there are a number of aspects that can be checked automatically without any rules. Examples are missing values or duplicates.

In the following, the categories of the horizontal dimension, presented in Section 3.1, are listed. The individual aspects of these categories are classified on the basis of the categories of the vertical dimension just presented.

### 3.2.1 Data loading

On the horizontal axis, the first category is data loading. Table 2 shows the aspects mentioned and describes how an evaluation can be done.

Tab. 2: Evaluation of data quality at the time of data loading

| Data quality aspect | Evaluation method   |  |  |  |
|---------------------|---|--|--|--|
|                     | Auto-<br>matic<br> | Rules<br> | Experts<br> | Ground<br>Truth<br> |
| Timeliness          | -   | (✓)  | ✓  | -  |
| Credibility         | -   | (✓)  | ✓  | -  |
| Relevance           | -   | (✓)  | ✓  | -  |

None of the aspects can be tested by automated evaluation without rules or domain knowledge. With rules, the following evaluations are conceivable: For timeliness, checks can be made according to the arrival time and intervals of the data. For credibility, a verification by

domain experts is necessary. In addition, checks according to the range of data or accepted values are conceivable. In terms of relevance, the assessment of whether data is suitable for a specific use case, primarily depends on the goal of the analysis or prediction. It must be evaluated by a domain expert. Information about the distribution of the data, warnings of protected features and fairness metrics can be supportive in the decision-making process. However, all these rules can only support the evaluation. The final assessment must be made by a domain expert. Ground truth cannot be detected for these aspects. For example, there is no ground truth that tells whether a data set is credible. Such aspects can only be assessed by domain experts. This is why ground truth cannot be used for an evaluation of these aspects.

### 3.2.2 Data preparation process

The goal of data preprocessing is to achieve the best possible data quality. As described, many different approaches and tools exist for this purpose. For better comparability, it must be possible to evaluate the results of such tools. The following metrics are suitable for this purpose, but also to subject the data to continuous evaluation. Thus, the quality of the data can be tracked at any time. Table 3 shows how the quality aspects can be evaluated in the context of the data preparation process.

All these aspects can be checked with ground truth, if available. If this is not available, the next step could be a manual verification by domain experts. If domain experts can not support, all these aspects could be checked using rules or external sources. The better the rules, the more accurate the assessment of the data quality. If no rules or external sources are available, only certain data quality aspects can be checked automatically. This includes the following aspects:

*Missing values* can be easily detected automatically. Rules are only needed here if missing values are encoded by specific values, such as -9999.

*Duplicates* can – to a certain degree – be checked automatically. At column level (*Uniqueness value violation*), duplicate detection is easy, but it is usually allowed for most columns that several rows have the same value. Further information is therefore needed to decide for which columns duplicates must not exist. On row level (*Redundancy about an entity*) it can be checked automatically if identical rows exist. For rows that are merely similar, however, further information is again required for evaluation.

*Outliers* can – to a certain extent – be detected automatically as well. A common approach is to apply the *three sigma rule*, which states that if the data is normally distributed, 99.7% of the values will be within three standard deviations of the mean [CBK09]. However, if the values of a column do not follow the normal distribution, this rule is not applicable. Furthermore, problems arise in this context when data changes and therefore the statistical

Tab. 3: Evaluation of data quality at the time of the data preparation process

| Data quality aspect                       | Evaluation method   |  |  |  |
|---|---|--|--|--|
|   | Auto-<br>matic<br> | Rules<br> | Experts<br> | Ground<br>Truth<br> |
| Missing Value                             | ✓   | ✓  | ✓  | ✓  |
| Syntax violation                          | -   | ✓  | ✓  | ✓  |
| Interval violation                        | -   | ✓  | ✓  | ✓  |
| Set violation                             | -   | ✓  | ✓  | ✓  |
| Misspelled error                          | -   | ✓  | ✓  | ✓  |
| Inadequate value to the attribute context | -   | ✓  | ✓  | ✓  |
| Value items beyond the attribute context  | -   | ✓  | ✓  | ✓  |
| Meaningless Value                         | -   | ✓  | ✓  | ✓  |
| Erroneous entry                           | -   | ✓  | ✓  | ✓  |
| Uniqueness value violation                | (✓)   | ✓  | ✓  | ✓  |
| Synonyms existence                        | -   | ✓  | ✓  | ✓  |
| Outlier                                   | (✓)   | ✓  | ✓  | ✓  |
| Missing Attribute                         | -   | ✓  | ✓  | ✓  |
| Semi-empty tuple                          | ✓   | ✓  | ✓  | ✓  |
| Inconsistency among attribute values      | -   | ✓  | ✓  | ✓  |
| Irrelevant observation                    | -   | ✓  | ✓  | ✓  |
| Redundancy about an entity                | ✓   | ✓  | ✓  | ✓  |
| Inconsistency about an entity             | -   | ✓  | ✓  | ✓  |
| Bias                                      | (✓)   | ✓  | ✓  | ✓  |
| Noise                                     | -   | ✓  | ✓  | ✓  |

parameters may need to be re-examined. For more extensive outlier detection, additional rules or sources are needed.

*Semi-empty tuples* can be checked automatically. As with missing values, rules are needed only in case empty fields are encoded by values such as -9999.

*Bias*: Already at the time of data loading, as described in Section 3.2.1, it should be checked whether the data is relevant. This also includes the analysis of whether the data is representative for the use case. However, as described in [SS20], a bias can also be introduced into the data by preprocessing. Therefore, as a first indicator, an automated check can be made to see if the distribution of the data changes significantly during preprocessing.

*All other aspects* cannot be checked automatically. Further information, rules or external sources are required for an evaluation.

The framework does not yet consider error hierarchies. For example, spelling errors may also lead to a set violation. After the spelling error has been corrected, there would possibly no longer be a set violation. In future work, this should be included in the framework.

### 3.2.3 Final evaluation

As already mentioned, all aspects specified in Section 3.2.2 should be checked again in the final evaluation. In addition, the presentation quality can also be evaluated. This is not possible without any rules or domain knowledge, as shown in Table 4.

Tab. 4: Evaluation of data quality for the final evaluation

| Data quality aspect  | Evaluation method   |   |   |   |
|----------------------|---|---|---|---|
|                      | Auto-<br>matic  | Rules   | Experts   | Ground<br>Truth   |
| Presentation Quality |  |  |  |  |
|                      | -   | (✓)   | ✓   | -   |

For an evaluation against rules, checks can be made based on given standards or specifications. A detailed evaluation can only be done by a domain expert. Ground truth does not exist in this case.

### 3.3 Measurement

In the following, we will look at how the metrics presented can be applied and how a measurement of quality can be made. Along the vertical dimension, evaluation becomes

more accurate once domain knowledge becomes available. Along the horizontal axis, a distinction is made depending on the time of data preparation.

### 3.3.1 Data loading

For all these criteria, it is only possible to check whether they are met or not. No percentage is given. For example, a data set may or may not be suitable for the use case. It is not checked whether it is 50% suitable. In future work, a more precise distinction could be made here.

### 3.3.2 Data preparation process

For each error listed in Table 1, the corresponding metric indicates whether that error occurs. Depending on the level of error classification, a distinction is made: In some cases, the percentage of errors per column is considered. In other cases, it is only checked whether the error occurs in the data (yes/no). The mapping depending on the level of error classification is shown in Table 5.

Tab. 5: Evaluation type per error classification level

| Level                                  | Evaluation Type   |
|--|---|
| An Attribute Value of a Single Tuple   | Percentage per column   |
| The Values of a Single Attribute       | Percentage per column<br>(except for <i>Missing Attribute</i> ) |
| The Attribute Values of a Single Tuple | Percentage per data set   |
| The Attribute Values of Several Tuples | Yes/no per data set   |

For the error types of the first level, *An Attribute Value of a Single Tuple*, the percentage per column can be measured in each case. The same applies to the second level, *The Values of a Single Attribute*. However, the error type *Missing Attribute* is an exception. Here, the percentage per data set must be applied. This is also considered for the third level, *The Attribute Values of a Single Tuple*. For the fourth level, a percentage can no longer be calculated. It is only checked whether the corresponding error type occurs in the data set or not.

An exception is when ground truth is available. In this case, metrics such as *precision* and *recall* can be used for evaluation.

### 3.3.3 Final evaluation

The same aspects apply to the final evaluation. However, for the examination of the presentation quality (as with the aspects at the time of data loading) it can only be stated whether it is provided or not.

**Example** Each quality criterion corresponding to the rows of tables 2, 3, and 4 is assigned a metric. Thus, the framework is easily extendable. The metric for *Missing Value* is described here as an example: The error type belongs to the first level *An Attribute Value of a Single Tuple* (see Table 1). Therefore, the metric is measured by the percentage per column (see Table 5). As can be seen in Table 3, an automatic verification is possible. Thus, the percentage of missing values per column in the data set can be calculated automatically. If there is a special encoding of missing values, further rules are necessary to be able to determine the percentage correctly. If these rules are not available, a domain expert can give the correct percentage. In case that ground truth is present, precision and recall can be applied for evaluation as described.

As a result of the evaluation, a corresponding report with all described metrics would be generated. Via color coding, the reliability of the results could be marked according to the vertical dimension. Thus, a detailed evaluation of data quality would be possible. In addition to validating individual data sets, different data sets can also be compared, for example to contrast the quality of serving and training data in context of machine learning.

## 4 Conclusion and Future Work

To measure data quality and evaluate data preparation tools, we created a framework of metrics. This can be used flexibly, depending on the extent to which ground truth is available or domain experts are available for verification. It also considers the time of the evaluation. Different aspects need to be checked at the beginning of data preprocessing rather than during the process itself. It was shown that domain knowledge is needed especially at the point of data loading and the initial investigation of various quality aspects, such as the usability of the data. As data preparation continues, more automation in evaluation is possible.

In the future, the presented framework will be made available to domain experts for an empirical validation. They should review the framework to ensure that all relevant data quality aspects for their domain and data engineering application are included in the classification. Beyond that, we would like to further develop this framework in future work. This includes further levels of the error classification presented as well as the consideration of error hierarchies. Moreover, only the metric for *Missing Value* was described as an example. The elaboration of the other metrics will be done accordingly to emphasize

practical relevance. In addition to the further development of the theoretical foundations, implementation also has to be carried out. In our vision, a framework like this must be an integral part of every data engineering pipeline to ensure data quality through data preparation.

## References

- [Ab16] Abedjan, Z. et al.: Detecting Data Errors: Where are we and what needs to be done? Proc. VLDB Endow. 9/12, pp. 993–1004, 2016, URL: <http://www.vldb.org/pvldb/vol9/p993-abedjan.pdf>.
- [BM11] Blake, R. H.; Mangiameli, P.: The Effects and Interactions of Data Quality and Problem Complexity on Classification. ACM J. Data Inf. Qual. 2/2, 8:1–8:28, 2011, URL: <https://doi.org/10.1145/1891879.1891881>.
- [CBK09] Chandola, V.; Banerjee, A.; Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. 41/3, 15:1–15:58, 2009, URL: <https://doi.org/10.1145/1541880.1541882>.
- [CZ15] Cai, L.; Zhu, Y.: The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. Data Sci. J. 14/, p. 2, 2015, URL: <https://doi.org/10.5334/dsj-2015-002>.
- [He19] Heidari, A. et al.: HoloDetect: Few-Shot Learning for Error Detection. In: Proceedings SIGMOD 2019. ACM, pp. 829–846, 2019, URL: <https://doi.org/10.1145/3299869.3319888>.
- [HH16] Heinrich, B.; Hristova, D.: A quantitative approach for modelling the influence of currency of information on decision-making under uncertainty. J. Decis. Syst. 25/1, pp. 16–41, 2016, URL: <https://doi.org/10.1080/12460125.2015.1080494>.
- [Ma19] Mahdavi, M. et al.: Raha: A Configuration-Free Error Detection System. In: Proceedings SIGMOD 2019. ACM, pp. 865–882, 2019, URL: <https://doi.org/10.1145/3299869.3324956>.
- [Pi20] Pitoura, E.: Social-minded Measures of Data Quality: Fairness, Diversity, and Lack of Bias. ACM J. Data Inf. Qual. 12/3, 12:1–12:8, 2020, URL: <https://dl.acm.org/doi/10.1145/3404193>.
- [Po17] Polyzotis, N.; Roy, S.; Whang, S. E.; Zinkevich, M.: Data Management Challenges in Production Machine Learning. In: Proceedings SIGMOD 2017. ACM, pp. 1723–1726, 2017, URL: <https://doi.org/10.1145/3035918.3054782>.
- [Re22] Restat, V. et al.: GouDa - Generation of universal Data Sets: Improving Analysis and Evaluation of Data Preparation Pipelines. In: Proceedings DEEM '22. ACM, 2:1–2:6, 2022, URL: <https://doi.org/10.1145/3533028.3533311>.
- [RKS22] Restat, V.; Klettke, M.; Störl, U.: Towards a Holistic Data Preparation Tool. In: Proceedings DATAPLAT '22. Vol. 3135, CEUR-WS.org, 2022, URL: [https://ceur-ws.org/Vol-3135/dataplat\\_short1.pdf](https://ceur-ws.org/Vol-3135/dataplat_short1.pdf).

- [Sc18] Schelter, S.; Lange, D.; Schmidt, P.; Celikel, M.; Bießmann, F.; Grafberger, A.: Automating Large-Scale Data Quality Verification. Proc. VLDB Endow./, pp. 1781–1794, 2018, URL: <http://www.vldb.org/pvldb/vol11/p1781-schelter.pdf>.
- [Si12] Sidi, F. et al.: Data quality: A survey of data quality dimensions. In: 2012 International Conference on Information Retrieval & Knowledge Management. IEEE, pp. 300–304, 2012, URL: <https://doi.org/10.1109/InfRKM.2012.6204995>.
- [SS20] Schelter, S.; Stoyanovich, J.: Taming technical bias in machine learning pipelines. IEEE Data Engineering Bulletin (Special Issue on Interdisciplinary Perspectives on Fairness and Artificial Intelligence Systems) 43/4, pp. 39–50, 2020.
- [Wi16] Wilkinson, M.D. et al.: The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data/, p. 160018, Mar. 2016, ISSN: 2052-4463, URL: <https://doi.org/10.1038/sdata.2016.18>.



# Recursive SQL and GPU-Support for In-Database Machine Learning

Maximilian E. Schüle<sup>1</sup>

**Abstract:** In machine learning, continuously retraining a model guarantees accurate predictions based on the latest data as training input. But to retrieve the latest data from a database, time-consuming extraction is necessary as database systems have rarely been used for operations such as matrix algebra and gradient descent. In this work, we demonstrate that SQL with recursive tables makes it possible to express a complete machine learning pipeline out of data preprocessing, model training and its validation. To facilitate the specification of loss functions, we extend the code-generating database system Umbra by an operator for automatic differentiation for use within recursive tables: With the loss function expressed in SQL as a lambda function, Umbra generates machine code for each partial derivative. We further use automatic differentiation for a dedicated gradient descent operator, which generates LLVM code to train a user-specified model on GPUs. We fine-tune GPU kernels at hardware level to allow a higher throughput and propose non-blocking synchronisation of multiple units. In our evaluation, automatic differentiation accelerated the runtime by the number of cached subexpressions compared to compiling each derivative separately. Our GPU kernels with independent models allowed maximal throughput even for small batch sizes, making machine learning pipelines within SQL more competitive.

## References

- [Sc22] Schüle, M. E.; Lang, H.; Springer, M.; Kemper, A.; Neumann, T.; Günemann, S.: Recursive SQL and GPU-support for in-database machine learning. *Distributed Parallel Databases* 40/2, pp. 205–259, 2022.

---

<sup>1</sup> University of Bamberg, maximilian.schuele@uni-bamberg.de



# VERIFAI - A Step Towards Evaluating the Responsibility of AI-Systems

Sabrina Göllner<sup>1</sup>, Marina Tropmann-Frick<sup>2</sup>

**Abstract:** This work represents the first step towards a unified framework for evaluating an AI system's responsibility by building a prototype application. The python based web-application uses several libraries for testing the fairness, robustness, privacy, and explainability of a machine learning model as well as the dataset which was used for training the model. The workflow of the prototype is tested and described using images of a healthcare dataset since healthcare represents an area where automatic decisions affect human lives, and building responsible AI in this area is therefore indispensable.

**Keywords:** Artificial Intelligence; Responsible AI; Privacy-preserving AI; Explainable AI; Ethical AI; Trustworthy AI

## 1 Introduction

This paper is based on the Structured Literature Review 'Aspects and Views on Responsible AI' presented at the *LOD conference 2022* [LO22] and on the follow-up paper which is currently in the writing process. The aforementioned papers conclude from the current state of the art that *Responsible AI* encompasses the aspects of 'security, privacy, ethics, explainability, human-centeredness, and trust'. The trust aspects are also up to the user's perception and human-centeredness requires a human-in-the-loop setting and will be part of our future work. Our first goal is to verify the security, privacy, ethics, and explainability within an AI system through different metrics in a single framework. Therefore we have created 'VERIFAI' (eValuating thE ResponsibIlity oF AI-systems), which is a first step towards putting this concept into practice. To the best of our current knowledge, there is no other framework that checks and evaluates multiple responsibility factors, so this is the novelty of the present work.

## 2 Implementation

This section is divided into two parts: the first part explains the selection of the data, and model architecture as well as the selection of toolkits for the evaluations and the second

---

<sup>1</sup> Hamburg University of Applied Sciences, Department of Computer Science, Berliner Tor 7, 20099 Hamburg, Germany [sabrina.goellner@haw-hamburg.de](mailto:sabrina.goellner@haw-hamburg.de)

<sup>2</sup> [marina.tropmann-frick@haw-hamburg.de](mailto:marina.tropmann-frick@haw-hamburg.de)

part consists of the presentation of the resulting web application based on an example walkthrough.

## 2.1 Dataset and model architecture

For the prototype implementation the healthcare dataset *HAM10000* [Ts18] was chosen because it satisfies two criteria: 1) it consists of dermatoscopic images from different populations including a representative collection of all important diagnostic categories in the realm of pigmented lesions and 2) because it consists not only of image data but also of metadata for the analysis.

The chosen model architecture for testing is *Xception* [Ch17], which is a network with a linear stack of depthwise separable convolution layers with residual connections. It achieved the best results on the dataset compared to other architectures.

### 2.1.1 Selection of toolkits

Since the project's goal was to verify the ethics, security, privacy, and explainability of both the data and model, the first step was to research state-of-the-art toolkits for testing. From the result of the libraries found, each of them could be classified into one of our four categories:

1. Evaluation of Explainability: Quantus [He22], IBM AI Explainability 360: [Ar19]
2. Evaluation of Ethics: Tensorflow Fairness Indicators [Te22], IBM AI Fairness 360 [Be18], Fairlearn [Bi20], Aequitas [Sa18], REVISE [Wa22], VISSL [Go21],
3. Evaluation of Security: IBM Adversarial Robustness 360 Toolkit [Ni18], Foolbox: [Ra20], Advbox [Go20], UnMask [Fr20]
4. Evaluation of Privacy: Privacy Meter [Sh22], IBM: differential privacy toolkit [Ho19], Tensorflow Privacy [ACP22]

Based on the features, metrics, quality, and usage limitations of the analyzed toolkits and libraries we came up with the following decisions for the prototype:

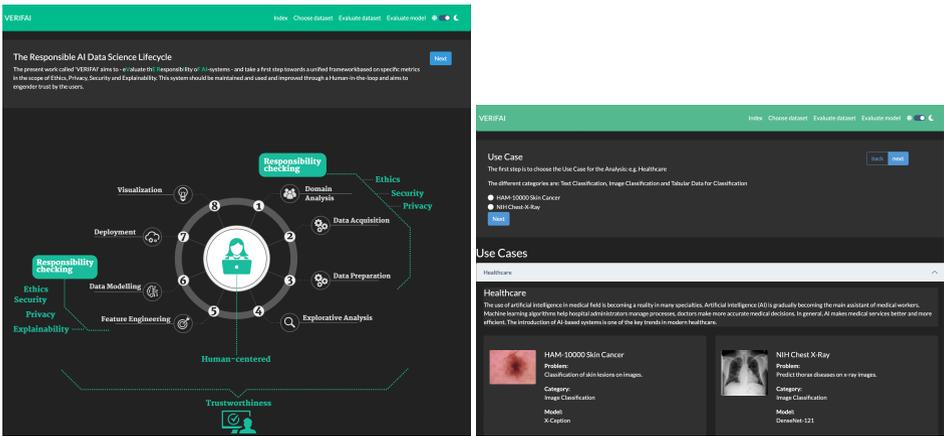
For the ethics/fairness evaluation of the model, there was, unfortunately, no suitable library that could handle medical image data properly, so the results were calculated without the usage of a toolkit, but with self-written python functions. The well-documented *Tensorflow privacy* was chosen for the privacy verification. For the security verification, the robustness test was performed using *Foolbox* because the calculation is reliable and fast, and the set of metrics can be extended in the future with others from the same library. In terms of explainability toolboxes, the choice fell on the *Quantus* toolbox because it supports evaluations of all kinds of neural networks and provides many different metrics to test

against that can be used for comparison in the future. To counteract confusion, the terms 'robustness' will be used instead of security and 'fairness' instead of ethics hereafter, as these are also referred to as such in the evaluations in this context.

## 2.2 Exemplary Walkthrough

This section shows the exemplary program flow based on an example with the presented data set using screenshots of the results and explanations.

### 2.2.1 The responsible data science lifecycle and the selection of the use case



(a) Screenshot: index / responsible data science lifecycle

(b) Screenshot: use cases

Fig. 1: Dataset

The index page, shown in figure 1a, is intended to introduce the systems' workflow to the user. It also offers a figure of the *Responsible Data Science Lifecycle*, which is the data science lifecycle extended through responsibility checks. At the top of the web page, there is an explanation of the current step, this continues throughout the application. Figure 1b is a screenshot of the step, where the user can choose from different use cases and corresponding data sets to run the evaluation on. In this case, we choose the HAM10000 image dataset with pigmented skin lesions which belongs to the healthcare use cases. By choosing the dataset we can go to the next step.

### 2.2.2 Exploratory fairness analysis of the dataset

In this step, we can analyze our dataset based on an exploratory fairness analysis.

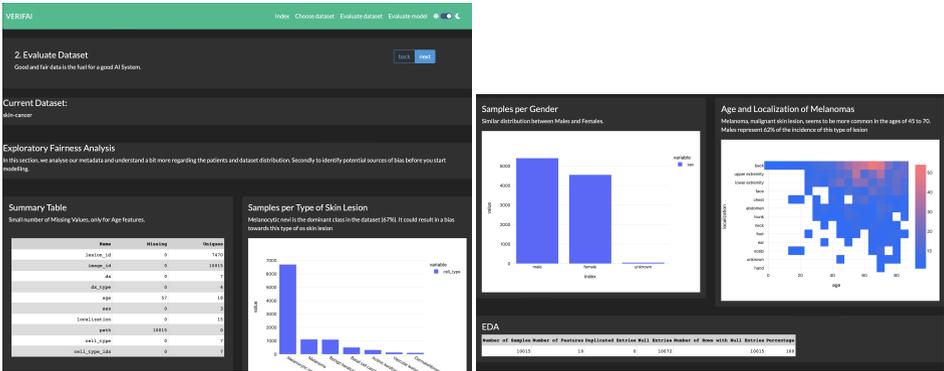


Fig. 2: Screenshot: exploratory fairness analysis of the dataset

Figure 2a and 2b display the data exploration for detecting potential biases to the user. For example, we can see that class *melanocytic nevi* is the dominant one in the dataset (67%). Using the data for modeling could result in a bias towards this type of skin lesion. This analysis is to help users detect such biases and prevent bad modeling.

### 2.2.3 Evaluate Model

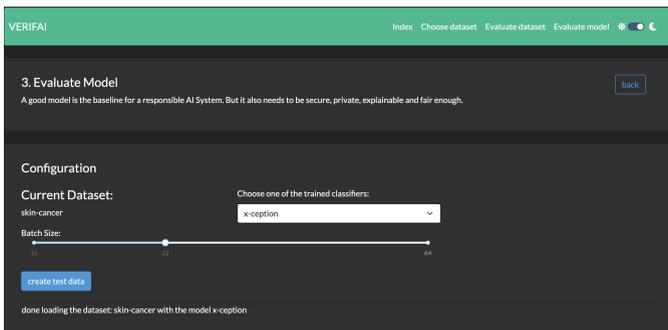


Fig. 3: Screenshot: load model and set configurations

In this step (see figure 3), the user can select the model and configuration for evaluation. The model in this example is an already trained *Xception* model. Choosing the batch size for the test dataset is also possible. The evaluations for robustness, fairness, privacy, and explainability are explained next.

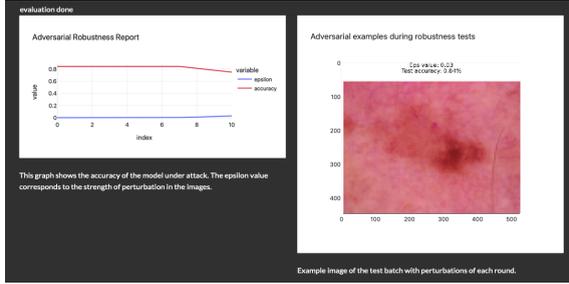


Fig. 4: Screenshot: evaluate model robustness (result)

**Robustness Evaluation** Figure 4 shows the results of testing if the model is robust against adversarial attacks with perturbed images, a metric which is called *adversarial robustness*. This is tested using the Projected Gradient Descent (PGD) attack. It attempts to find the perturbation that maximizes the loss of a model on a particular input while keeping the size of the perturbation smaller than a specified amount referred to as epsilon ( $\epsilon$ ) while  $\max. \epsilon = 0.3$ , as this is used for benchmarking in the *RobustBench* [Cr20]). Each round the  $\epsilon$  is increasing and the robustness score is measured as shown in the plots: starting from 82%, the model has still an accuracy of 72% in the last round, using a perturbation of  $\epsilon = 0.3$ , which is still a good accuracy. The image on the right is an example image from the test batch with the maximum perturbations ( $\epsilon = 0.3$ ) added.

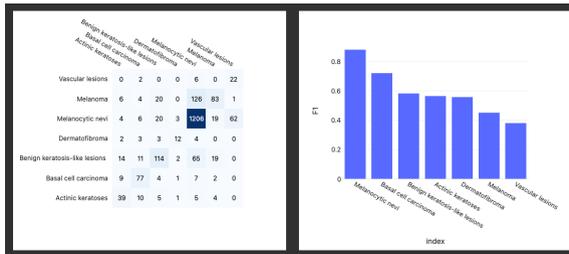


Fig. 5: Screenshot: evaluate fairness (result)

**Fairness Evaluation** Figure 5 shows the correctly and incorrectly classified images through a confusion matrix. We can see, that the tested model is very biased in the direction of the class *melanocytic nevi*. The reason for this result is probably because we already had an imbalanced dataset before. The second plot is the F1-score for the different classes, which is suitable for imbalanced data. Because we have a bias in the model the end results are not good enough for a good fairness score.

**Privacy Evaluation** Figure 6 shows the check of the privacy leakage through a *membership inference attack*, which tries to find out if specific examples were in the training set (see fig. 6a). The results show that the membership inference attack was successful but had only an

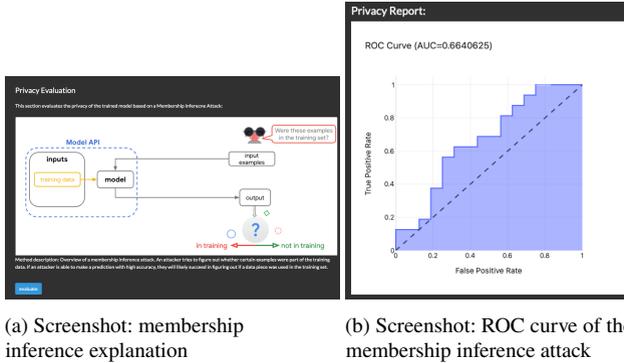


Fig. 6: Screenshots: Evaluate model privacy (results)

AUC of 66% (see fig. 6b). This means, that the privacy leakage was only satisfactory for the attacker, which is better for the model’s privacy score, we can therefore determine that the leak of information was only moderate in this case.

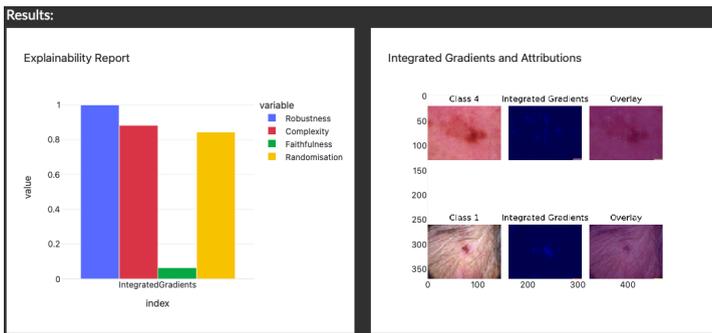


Fig. 7: Screenshot: Evaluate Model Explainability

**Explainability Evaluation** Figure 7 displays the quantitative evaluation of the model’s explanation in combination with the chosen explainability method. We chose the suitable XAI-Method called *Integrated Gradients*. The used metrics are: *Robustness*, which measures the probability that the inputs with the same explanation have the same prediction label [Ye19], *Complexity* that measures if only highly attributed features are truly predictive of the model output [Ch18], *Faithfulness* which iteratively replaces a random subset of given attributions with a baseline value and then measures the correlation between the sum of this attribution subset and the difference in function output [BWM20], and *Randomisation*, which computes for the distance between the original explanation and the explanation for a random other class [SGL20]. In the barplot on the left, we can see that Robustness, Complexity, and Randomisation scored well with relatively high percentages, but Faithfulness did not. All four scores contribute with equal weighting to the final result and their scores were therefore

averaged. In the figure next to the bar chart, we can see also examples of images from the test batch, and the corresponding explanation. This is so that the user can also get an idea of whether the explanation is good enough or not.



Fig. 8: Screenshot: Responsibility Evaluation

**Responsibility Evaluation** Finally, the *Responsibility Evaluation* in figure 8 summarizes the calculated scores using the proposed metrics and highlights them with different colors according to their scores. The rating was calculated as follows: A 'perfect' model would score full points in every aspect, which equals 10 points. In our test case, the security evaluation was tested with a good result (8/10) as well as privacy (6/10), while the other metrics fairness and explainability (6/10) show still some weaknesses and achieved therefore moderate scores. The worst result was the fairness score of the model (5/10) because of the bias. Thus, our model achieved a final score of 62.5% (25/40) with the metrics currently implemented.

### 3 Open challenges and future work

In this work, we created a prototype implementation of VERIFAI, an application for evaluating an AI system's responsibility based on several aspects. In the present prototype, we used a healthcare dataset. The tool can evaluate the dataset for fairness and a trained machine learning model for fairness, privacy leakage, adversarial robustness, and explainability using a variety of state-of-the-art metrics. Even though this work only covers a limited number of metrics so far, it is a good basis for future work. The following extensions are planned for future work: We will add more data sets belonging to different suitable scenarios, different machine learning models for each scenario, extend the set of metrics for each category, choose between selectable or auto-selection of the right metrics for the given problem, selectable target user, selectable focus for which aspect is most important for the target user, the tolerance level for each aspect, suggestions for mitigations, evaluation of trustworthiness and human-in-the-loop aspects. We are also working on making VERIFAI as transparent as possible for the users for helping to create more responsible AI systems.

## Bibliography

- [ACP22] Andrew, Galen; Chien, Steve; Papernot, Nicolas; , Tensor Flow Privacy. <https://github.com/tensorflow/privacy>, 2022.
- [Ar19] Arya, Vijay; Bellamy, Rachel KE; Chen, Pin-Yu; Dhurandhar, Amit; Hind, Michael; Hoffman, Samuel C; Houde, Stephanie; Liao, Q Vera; Luss, Ronny; Mojsilović, Aleksandra et al.: One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques. arXiv preprint arXiv:1909.03012, 2019.
- [Be18] Bellamy, Rachel K. E.; Dey, Kuntal; Hind, Michael; Hoffman, Samuel C.; Houde, Stephanie; Kannan, Kalapriya; Lohia, Pranay; Martino, Jacquelyn; Mehta, Sameep; Mojsilovic, Aleksandra; Nagar, Seema; Ramamurthy, Karthikeyan Natesan; Richards, John; Saha, Diptikalyan; Sattigeri, Prasanna; Singh, Moninder; Varshney, Kush R.; Zhang, Yunfeng: AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. October 2018.
- [Bi20] Bird, Sarah; Dudík, Miro; Edgar, Richard; Horn, Brandon; Lutz, Roman; Milan, Vanessa; Sameki, Mehrnoosh; Wallach, Hanna; Walker, Kathleen: Fairlearn: A toolkit for assessing and improving fairness in AI. Technical Report MSR-TR-2020-32, Microsoft, May 2020.
- [BWM20] Bhatt, Umang; Weller, Adrian; Moura, José MF: Evaluating and aggregating feature-based model explanations. arXiv preprint arXiv:2005.00631, 2020.
- [Ch17] Chollet, Francois: Xception: Deep Learning With Depthwise Separable Convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). July 2017.
- [Ch18] Chalasani, P; Chen, J; Chowdhury, AR; Jha, S; Wu, X: Concise explanations of neural networks using adversarial training. arXiv arXiv-1810. arXiv preprint arXiv:1810.06583, 2018.
- [Cr20] Croce, Francesco; Andriushchenko, Maksym; Sehwal, Vikash; Debenedetti, Edoardo; Flammarion, Nicolas; Chiang, Mung; Mittal, Prateek; Hein, Matthias: Robustbench: a standardized adversarial robustness benchmark. arXiv preprint arXiv:2010.09670, 2020.
- [Fr20] Freitas, Scott; Chen, Shang-Tse; Wang, Zijie J; Chau, Duen Horng: Unmask: Adversarial detection and defense through robust feature alignment. In: 2020 IEEE International Conference on Big Data (Big Data). IEEE, pp. 1081–1088, 2020.
- [Go20] Goodman, Dou; Xin, Hao; Yang, Wang; Yuesheng, Wu; Junfeng, Xiong; Huan, Zhang: Advbox: a toolbox to generate adversarial examples that fool neural networks. 2020.
- [Go21] Goyal, Priya; Duval, Quentin; Reizenstein, Jeremy; Leavitt, Matthew; Xu, Min; Lefaudeux, Benjamin; Singh, Mannat; Reis, Vinicius; Caron, Mathilde; Bojanowski, Piotr; Joulin, Armand; Misra, Ishan; , VISSL. <https://github.com/facebookresearch/vissl>, 2021.
- [He22] Hedström, Anna; Weber, Leander; Bareeva, Dilyara; Motzkus, Franz; Samek, Wojciech; Lapuschkin, Sebastian; Höhne, Marina M.-C.: Quantus: An Explainable AI Toolkit for Responsible Evaluation of Neural Network Explanations. 2022.
- [Ho19] Holohan, Naoise; Braghin, Stefano; Mac Aonghusa, Pól; Levacher, Killian: Diffprivlib: the IBM differential privacy library. ArXiv e-prints, 1907.02444 [cs.CR], July 2019.
- [LO22] LOD Conference: . <https://lod2022.icas.cc/program/>, 2022.

- [Ni18] Nicolae, Maria-Irina; Sinn, Mathieu; Tran, Minh Ngoc; Buesser, Beat; Rawat, Ambrish; Wistuba, Martin; Zantedeschi, Valentina; Baracaldo, Nathalie; Chen, Bryant; Ludwig, Heiko et al.: Adversarial Robustness Toolbox v1. 0.0. arXiv preprint arXiv:1807.01069, 2018.
- [Ra20] Rauber, Jonas; Zimmermann, Roland; Bethge, Matthias; Brendel, Wieland: Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. *Journal of Open Source Software*, 5(53):2607, 2020.
- [Sa18] Saleiro, Pedro; Kuester, Benedict; Stevens, Abby; Anisfeld, Ari; Hinkson, Loren; London, Jesse; Ghani, Rayid: Aequitas: A Bias and Fairness Audit Toolkit. arXiv preprint arXiv:1811.05577, 2018.
- [SGL20] Sixt, Leon; Granz, Maximilian; Landgraf, Tim: When explanations lie: Why many modified bp attributions fail. In: *International Conference on Machine Learning*. PMLR, pp. 9046–9057, 2020.
- [Sh22] Shokri, Reza: ML Privacy Meter: A Tool to Quantify Information Leakage through Machine Learning Models. 2022.
- [Te22] Tensorflow: , Tensor Flow Fairness Indicators. <https://github.com/tensorflow/fairness-indicators>, 2022.
- [Ts18] Tschandl, Philipp: The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. 2018.
- [Wa22] Wang, Angelina; Liu, Alexander; Zhang, Ryan; Kleiman, Anat; Kim, Leslie; Zhao, Dora; Shirai, Iroha; Narayanan, Arvind; Russakovsky, Olga: REVISE: A Tool for Measuring and Mitigating Bias in Visual Datasets. *International Journal of Computer Vision (IJCV)*, 2022.
- [Ye19] Yeh, Chih-Kuan; Hsieh, Cheng-Yu; Suggala, Arun; Inouye, David I; Ravikumar, Pradeep K: On the (in) fidelity and sensitivity of explanations. *Advances in Neural Information Processing Systems*, 32, 2019.



# Workload Prediction for IoT Data Management Systems

David Burrell<sup>1</sup>, Xenofon Chatziliadis<sup>2</sup>, Eleni Tzirita Zacharatou<sup>3</sup>, Steffen Zeuch<sup>4</sup>, Volker Markl<sup>5, 6</sup>

**Abstract:** The Internet of Things (IoT) is an emerging technology that allows numerous devices, potentially spread over a large geographical area, to collect and collectively process data from high-speed data streams. To that end, specialized IoT data management systems (IoTDMSs) have emerged. One challenge in those systems is the collection of different metrics from devices in a central location for analysis. This analysis allows IoTDMSs to maintain an overview of the workload on different devices and to optimize their processing. However, as an IoT network comprises of many heterogeneous devices with low computation resources and limited bandwidth, collecting and sending workload metrics can cause increased latency in data processing tasks across the network.

In this ongoing work, we present an approach to avoid unnecessary transmission of workload metrics by predicting CPU, memory, and network usage using machine learning (ML). Specifically, we demonstrate the performance of two ML models, linear regression and Long Short-Term Memory (LSTM) neural network, and show the features that we explored to train these models. This work is part of an ongoing research to develop a monitoring tool for our new IoTDMS named NebulaStream.

**Keywords:** Internet of Things; stream processing; machine learning; workload prediction

## 1 Introduction

The Internet of Things (IoT) describes a distributed system in which a large number of devices with sensing or processing capabilities communicate with each other [PPS16]. The IoT enables new possibilities for applications that have led to advances in many fields, including healthcare [Ab19], disaster management [OTM21], and smart cities [Mo21].

---

<sup>1</sup> TU Berlin, Database Systems and Information Management, Einsteinufer 17, 10587 Berlin, Germany d.burrell736@gmail.com

<sup>2</sup> TU Berlin, Database Systems and Information Management, Einsteinufer 17, 10587 Berlin, Germany x.chatziliadis@tu-berlin.de

<sup>3</sup> IT University of Copenhagen, Data-intensive Systems and Applications, Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark elza@itu.dk

<sup>4</sup> German Research Center for Artificial Intelligence (DFKI), Alt-Moabit 91c, 10559 Berlin, Germany steffen.zeuch@dfki.de

<sup>5</sup> TU Berlin, Database Systems and Information Management, Einsteinufer 17, 10587 Berlin, Germany volker.markl@tu-berlin.de

<sup>6</sup> German Research Center for Artificial Intelligence (DFKI), Alt-Moabit 91c, 10559 Berlin, Germany Volker.Markl@dfki.de

To process data in real time in such a distributed environment, stream processing engines (SPEs) are often used. SPEs enable the continuous execution of user-defined queries on data streams to extract useful information. SPEs are designed to execute these queries across distributed devices optimally, in terms of the placement of the data processing operations and the management of the information flow. However, SPEs are designed for the cloud, which has significantly different characteristics compared to the IoT.

For system-internal decision-making processes, such as load balancing and query optimization, existing SPEs often require performance metrics from their topology. In an IoT topology, the collection and sending of these metrics can be detrimental to the overall data processing efficiency. Many devices have few computational resources, and the collection of workload metrics leads to reduced processing performance [Ch21, CFSF20]. Additionally, the collection process affects the network, since it requires a large amount of bandwidth to send the collected metrics and therefore introduces additional latency [SJL18]. Finally, the additional processing and network efforts required for the metric collection process reduce the operational lifetime of battery-powered devices [Ma05].

To mitigate these issues, it would be beneficial for an SPE to be able to infer workload metric values and avoid the need to collect these metrics continuously. This would reduce the load on the workers themselves as well as the volume of data being sent through the network, thus decreasing the response time for user queries. In this work, we show our current progress in testing the suitability of machine learning techniques to predict the workloads of the NebulaStream (NES) IoT DMS [Ze20a, Ze20b]. Using a small-scale lab topology with five devices, we have trained a linear regression and an LSTM model to predict the CPU, memory, and network utilization of different query workloads. We show that for all workloads, our linear regression models significantly outperform the LSTMs with a PRED 25 score (i.e., percentage of predicted values that are within 25% of the actual value) between 56% and 86%.

The remainder of this paper is structured as follows: In Section 2, we analyze the different workloads and features for our ML models. In Section 3, we evaluate the suitability of linear regression and LSTM to predict CPU, memory, and network utilization. We conclude our work in Section 4 and discuss future work directions.

## 2 Methodology

Data collection and data engineering steps are of essential importance to produce adequate training data sets for accurate ML models. To this end, we create an emulated NES topology where we execute specific queries on predefined data to generate workload information. During the execution of these queries, we collect metrics on CPU, memory, and network utilization. We collect these metrics along with additional features about the topology that can be used to train our ML models. In the remainder of this section, we describe in detail

how we generate different workloads in NES and the features we investigate to train our ML models.

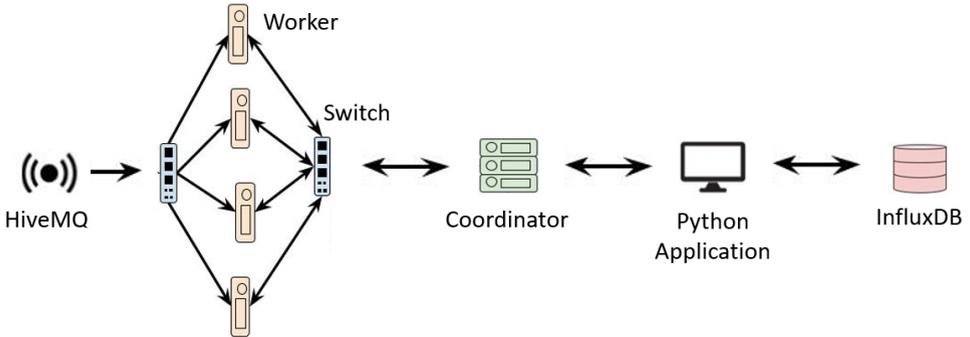


Fig. 1: Overview of the emulated topology in NebulaStream.

## 2.1 Workload Generation

For generating training load data we create a controlled NebulaStream lab topology with Docker containers [Me14]. Communication between containers is established using the Containernet network emulation [PKvR16]. In total, the topology consists of one coordinator, four workers, and an MQTT broker (cf. Fig. 1). The MQTT broker (HiveMQ) has four topics, one for each worker to subscribe to and where data is pushed in regular intervals. The data that is pushed to the workers is generated using the humidity and temperature sensors data set of the UCI Machine Learning Repository [Hu16]. The workers are used to process and send the data to the coordinator. In order to represent the heterogeneity of the IoT environment in our lab topology, we assign each worker different resources in terms of available memory, memory swap limit, and CPU shares. Regarding the distribution of resources, worker number zero had the fewest available resources with a memory limit of 10Mb, a memory swap limit of 30Mb, and a CPU share of 20%. For each worker, we incrementally increased then the resources, such that worker number three has 40Mb memory limit, 120Mb memory swap limit, and 80% CPU shares.

The coordinator is the central component of NebulaStream, which is responsible for processing user requests, scheduling queries, and managing the life cycle of running queries. We used the coordinator as a sink for our streaming queries and an endpoint for our queries. To generate different load types, we deployed four queries to NES, 1) a select-all query, 2) a projection, 3) a filter with a selectivity of 0.77, and 4) a user-defined function with the map operation. These queries were deployed from our Python application via the REST interface of NES. Workload metrics were collected through the NES monitoring API and the docker metrics API while running the queries. For analysis and feature exploration, we stored all workload metrics in the InfluxDB key-value store.

## 2.2 Feature Selection

For training accurate ML models, we require feature variables that have some relationship with the target variables, i.e., the workload metrics. In general, the stronger the relationship between the features and the target, the higher the model’s prediction accuracy. We have analyzed two different types of features.

The first type are static features, where the value does not change over time. The static features collected from the emulated topology are: the worker resource limits, the tuples per second received at the workers, the executed queries, and their operators. The second type of features are time-series features that we divide into two classes. The first class belongs to the independent variables that are recorded separately to the workload metric streams. These variables are the number of nanoseconds since the query began, the data received at the coordinator from each worker, and the coordinator’s CPU usage. The second set of time-series features are the metric streams of the workloads.

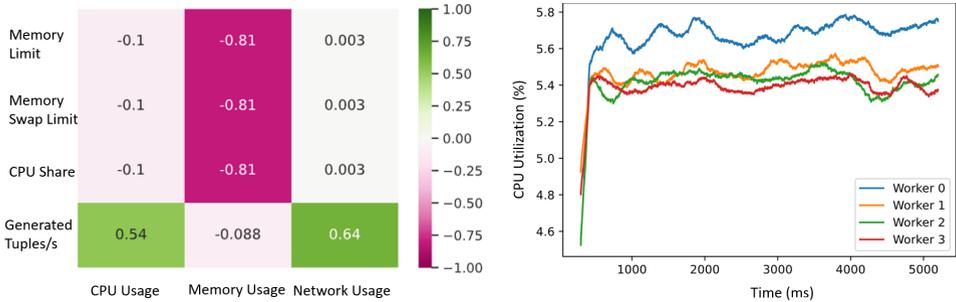


Fig. 2: (left) Heatmap of the SRCC between the static variables and the workload metrics. (right) The moving average of the CPU usage during the execution of the  $\beta$ select all"query.

To identify important candidates, we use Pearson and Spearman Rank Correlation Coefficient (SRCC) to determine if a relationship exists between potential features and our target variables CPU, memory, and network usage. Due to space limitations, we highlight here the most influential features, which can be seen based on SRCC between the resource limits (i.e., memory, memory swap, and CPU share) and memory usage, with a particularly strong negative correlation (cf. Fig. 2 (left)). The reason is that workers with the fewest resources available to process the arriving data are required to store the data for a longer period before the data can be sent to the coordinator.

By looking at the moving average of the values over time, it is possible to see the effect of having different percentages of CPU for each worker (cf. Fig. 2 (right)). For the moving average, we use a window size of 300ms and a sampling period of 20ms. It can be seen that worker number zero has the most CPU usage due to having the smallest share of CPU and requiring more time to complete tasks, while the other workers (which were closer together in value) followed in order of increasing percentage of CPU share.

In summary, the patterns we observe via SRCC and the moving average plot confirm that available resources on the workers have a huge effect on the workload and are thus important features that should be taken into account when training an ML model.

### 3 Evaluation

For each load metric, i.e., CPU, memory, and network, we train a linear regression and LSTM model, which results in six models in total. We evaluate the models using the PRED 25 score, which measures the percentage of predicted values that are within 25 percent of the actual value (cf. Table 1). The models are trained on 80 multivariant time series with 246.883 different samples, and for validation on 246.883 samples. Overall, the linear regression model (LR) is more accurate at predicting the three workload metrics than the LSTM model, which is similar to the results reported by [AB16]. In the remainder of this section, we will give a more detailed discussion of our results and findings.

| Query Type        | CPU Model |             | Memory Model |             | Network Model |             |
|-------------------|-----------|-------------|--------------|-------------|---------------|-------------|
|                   | <i>LR</i> | <i>LSTM</i> | <i>LR</i>    | <i>LSTM</i> | <i>LR</i>     | <i>LSTM</i> |
| <b>Select all</b> | 98.53     | 87.98       | 0.15         | 57.11       | 89.15         | 16.68       |
| <b>Projection</b> | 29.00     | 20.65       | 69.13        | 42.59       | 83.23         | 15.39       |
| <b>Filter</b>     | 94.94     | 39.83       | 76.59        | 43.23       | 89.26         | 28.17       |
| <b>Map</b>        | 49.20     | 39.45       | 89.85        | 36.94       | 91.97         | 22.80       |
| <b>Total Mean</b> | 73.07     | 50.32       | 58.87        | 46.13       | 86.34         | 28.55       |

Tab. 1: PRED 25 score for each of the machine learning models.

#### 3.1 CPU Model

The linear regression CPU model achieves, on average, for all query types a PRED 25 score of 73.07%, which significantly outperforms the LSTM model that reaches only 50.32%. Looking at the scores of individual query types, we can see that the linear regression performs extremely well for the select all and filter query type with a score over 90%. On the contrary, for the projection and map query, our models achieve a score below 50%. For the latter our models had captured in general the trends of the curves. However, they predicted incorrectly the starting values which consequently lead to wrong forecasts of the subsequent values.

### 3.2 Memory Model

For the memory model we reach with the linear regression in total a mean score of around 59% and the LSTM around 46%. Looking at the different query types individually we can see, on the one hand, that the performance of the select all query is the biggest detractor for the linear regression model with a score below 1%. During the execution of the projection, filter and map query, we could observe often spikes in the curves regarding memory consumption. However, for the select all query a spiking of memory consumption does not happen, as the internal memory management of NES is keeping for that particular case always the same amount of data tuples in memory. We believe that the performance of the linear regression can be improved here by adding more information about the query type to the ML model. For the projection, filter and map query, on the other hand, the linear regression achieved a performance of ca. 70% up to 90%. The LSTM model performed for the prediction of memory utilization again worse for all query types with a score between 37% and 57%.

### 3.3 Network Model

The linear regression model is performing consistently well for all different query types with a PRED 25 score between 83% and 92% and an average score of 86.34%. The LSTM on the contrary reaches only scores between ca. 15% and 28%. During training, we had observed that the LSTM model was having problems with over fitting, as the prediction performance on the training data set was always larger than 90%. In future work, we will further investigate the reasons why the LSTM was over fitting that extremely for predicting network utilization.

## 4 Conclusion and Future Work

This paper explores the applicability of ML approaches to predict workload characteristics in streaming IoT environments. Our early results show that for a particular small-scale lab topology, workloads can be estimated with a PRED 25 score of up to 86% by using a linear regression model. However, to completely avoid direct value collection from the devices in the topology, further work is needed to increase the prediction accuracy. In future work, we plan to incorporate topological traits and create workload data on the basis of more complex queries. In our current work, we tested our models only against a single data set that was generated on a static lab topology. Additionally, we plan to integrate our approach of workload prediction to NebulaStream, in order to evaluate the general benefits.

Finally, we also plan to investigate different query optimization and operator placement strategies. By examining the operators that are deployed on each worker we hope to be able to identify a pattern in which cases the static and time-series characteristics can replace the continuous collection of performance measures and why in other cases this does not work.

## Acknowledgments

This work has received funding by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A, 01IS18037A) and from the European Union's H2020 research and innovation programme under grant agreement No. 957286.

## References

- [AB16] Ajila, Samuel; Bankole, Akindele: Using Machine Learning Algorithms for Cloud Client Prediction Models in a Web VM Resource Provisioning Environment. *Transactions on Machine Learning and Artificial Intelligence*, 2016.
- [Ab19] Abdellatif, Alaa; Mohamed, Amr; Chiasserini, Carla-Fabiana; Tlili, Mounira; Erbad, Aiman: Edge Computing For Smart Health: Context-aware Approaches, Opportunities, and Challenges. *IEEE Network*, 2019.
- [CFSF20] Cid-Fuentes, Javier Álvarez; Szabo, Claudia; Falkner, Katrina E.: Adaptive Performance Anomaly Detection in Distributed Systems Using Online SVMs. *IEEE Transactions on Dependable and Secure Computing*, 2020.
- [Ch21] Chatziliadis, Xenofon; Tzirita Zacharatou, Eleni; Zeuch, Steffen; Markl, Volker: Monitoring of stream processing engines beyond the cloud: an overview. *Open Journal of Internet Of Things (OJIOT)*, 2021.
- [Hu16] Huerta, Ramon; Mosqueiro, Thiago; Fonllosa, Jordi; Rulkov, Nikolai F; Rodriguez-Lujan, Irene: Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 2016.
- [Ma05] Madden, Samuel R.; Franklin, Michael J.; Hellerstein, Joseph M.; Hong, Wei: TinyDB: An acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 2005.
- [Me14] Merkel, Dirk: Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014.
- [Mo21] Moreno-Bernal, Pedro; Cervantes-Salazar, Carlos Alan; Nesmachnow, Sergio; Hurtado-Ramírez, Juan Manuel; Hernández-Aguilar, José Alberto: Open-Source Big Data Platform for Real-Time Geolocation in Smart Cities. *Ibero-American Congress of Smart Cities*, 2021.
- [OTM21] Ouro Paz, Elena Beatriz; Tzirita Zacharatou, Eleni; Markl, Volker: Towards Resilient Data Management for the Internet of Moving Things. In: *Datenbanksysteme für Business, Technologie und Web (BTW)*. volume P-311 of LNI, pp. 279–301, 2021.
- [PKvR16] Peuster, M.; Karl, H.; van Rossem, S.: MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments. *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

- [PPS16] Patel, Keyur K; Patel, Sunil M; Scholar, P: Internet of things-IOT: definition, characteristics, architecture, enabling technologies, application & future challenges. International journal of engineering science and computing, 2016.
- [SJL18] Son, Yunsik; Jeong, Junho; Lee, YangSun: An Adaptive Offloading Method for an IoT-Cloud Converged Virtual Machine System Using a Hybrid Deep Neural Network. Sustainability, 2018.
- [Ze20a] Zeuch, Steffen; Chaudhary, Ankit; Monte, Bonaventura Del; Gavriilidis, Haralampos; Giouroukis, Dimitrios; Grulich, Philipp M.; Bress, Sebastian; Traub, Jonas; Markl, Volker: The NebulaStream Platform: Data and Application Management for the Internet of Things. CIDR, 2020.
- [Ze20b] Zeuch, Steffen; Tzirita Zacharatou, Eleni; Zhang, Shuhao; Chatziliadis, Xenofon; Chaudhary, Ankit; Del Monte, Bonaventura; Giouroukis, Dimitrios; Grulich, Philipp M; Ziehn, Ariane; Mark, Volker: NebulaStream: Complex analytics beyond the cloud. The International Workshop on Very Large Internet of Things (VLIoT), 2020.

# A Provenance Management Framework for Knowledge Graph Generation in a Web Portal

Erik Kleinstauber,<sup>1</sup> Samira Babalou,<sup>2</sup> Birgitta König-Ries<sup>3</sup>

**Abstract:** Knowledge Graphs (KGs) are the semantic backbone for a wide variety of applications in different domains. In recent years, different web portals providing relevant functionalities for managing KGs have been proposed. An important functionality of such portals is provenance data management of the KG generation process. Capturing, storing, and accessing provenance data efficiently are complex problems. Solutions to these problems vary widely depending on many factors like the computational environment, computational methods, desired provenance granularity. In this paper, we present one possible solution: a new framework to capture coarse-grained workflow provenance of KGs during creation in a web portal. We capture the necessary information of the KG generation process and store and retrieve the provenance data using standard functionalities of relational databases. Our captured workflow can be rerun over the same or different input source data. With this, the framework can support four different applications of provenance data: (i) reproduce the KG, (ii) create a new KG with an existing workflow, (iii) undo the executed tools and adapt the provenance data accordingly, and (iv) retrieve the provenance data of a KG.

**Keywords:** Semantic Web; Knowledge Graph; Knowledge Graph Platform; Provenance Tracking; Reproducibility

## 1 Introduction

Knowledge Graphs (KGs) are graph-structured knowledge bases that store factual information about a particular domain in the form of relationships between entities. They are the semantic backbone for a wide variety of applications. This brings the need to support their management. In consequence, different KG management platforms have been suggested for both scientific and commercial applications [Sy22, SDA20, Ha19, Be20]. Such platforms mostly cover the whole lifecycle of KG application and include relevant services or functionalities for creating, using, and further management of KGs. The KG generation process (see Figure 1, top) can be modeled as an ordered execution of tools to transform source data into a data graph. Ideally, during this creation process, detailed provenance information should be

---

<sup>1</sup> Heinz-Nixdorf Chair for Distributed Information Systems, Institute for Computer Science, Friedrich Schiller University Jena, Germany erik.kleinstauber@uni.jena.de

<sup>2</sup> Heinz-Nixdorf Chair for Distributed Information Systems, Institute for Computer Science, Friedrich Schiller University Jena, Germany; German Center for Integrative Biodiversity Research (iDiv), Halle-Jena-Leipzig samira.babalou@uni.jena.de

<sup>3</sup> Heinz-Nixdorf Chair for Distributed Information Systems, Institute for Computer Science, Friedrich Schiller University Jena, Germany; German Center for Integrative Biodiversity Research (iDiv), Halle-Jena-Leipzig; Michael-Stifel-Center for Data-Driven and Simulation Science, Jena, Germany birgitta.koenig-ries@uni-jena.de

captured and subsequently managed. With this, on the one hand, it is possible to track where information in the KG stems from. This increases trust in the information provided and supports open science principles [SKR22]. On the other hand, provenance information enables rerunning the creation process to check for reproducibility and/or to create an updated version of a KG. Both are important factors in increasing KGs usage. Capturing, storing, and accessing provenance data are complex problems. Solutions to them vary widely depending on many factors like the computational environment, computational methods, desired provenance granularity, and much more. The decision on a provenance solution depends on interests, needs, and expectations of the developers or potential users, and heavily on the domain of applications [PRSA18]. One solution is to integrate a computational task into an environment capable of capturing provenance of the results of these tasks.

In a KG-generation web portal with user interaction, it is necessary to consider dependencies, restrictions, security, and fault tolerance issues of different tools during KG generation process. Third-party tools need to be securely connected, safely executed, and carefully monitored. It is also important to embed these tools in a way that they do not affect each other (e.g., writing files or memory usage). Moreover, requirements on the provenance solution for the web portal might change during development, thus a highly flexible architecture is required. Existing provenance systems (cf. the reviewed systems in [PRSA18]) tend to have constraints on programming languages and on domains. Integrating third-party tools into a computational notebook can be a difficult task depending on the complexity of the tool. Additionally, connecting such a provenance system or computational notebook to the web portal's functionalities adds an extra layer of complexity if possible at all.

In this paper, we present our solution to providing users with all necessary information (so-called provenance data) about the KG generation process in a web portal. We present the provenance data of KG generation process as a workflow. It holds information about all executed tools and necessary inputs and outputs during KG generation. We propose a framework that can capture the coarse-grained workflow provenance of generated KGs in a web portal. This framework is customized in our studied platform, but it can be adopted for any other platform. We capture the necessary information about the user, source data, intermediate results, and executed tools during the KG generation process. We ensure that the data about the computational tasks needed to create the KG are stored in a reusable workflow associated to the KG. We retrieve the captured provenance data accordingly to provide user convenience and better insight into the KG generation process. Our captured workflow can be rerun over the same or different source data. We show different applications of the provenance data. Moreover, we show how the workflow in our framework can be mapped to the W3C PROV ontology [Le13].

The rest of the paper is organized as follows. Section 2 presents preliminaries along with literature review. Section 3 shows our proposed framework, followed by implementation detail in Section 4. We conclude the paper in Section 6.

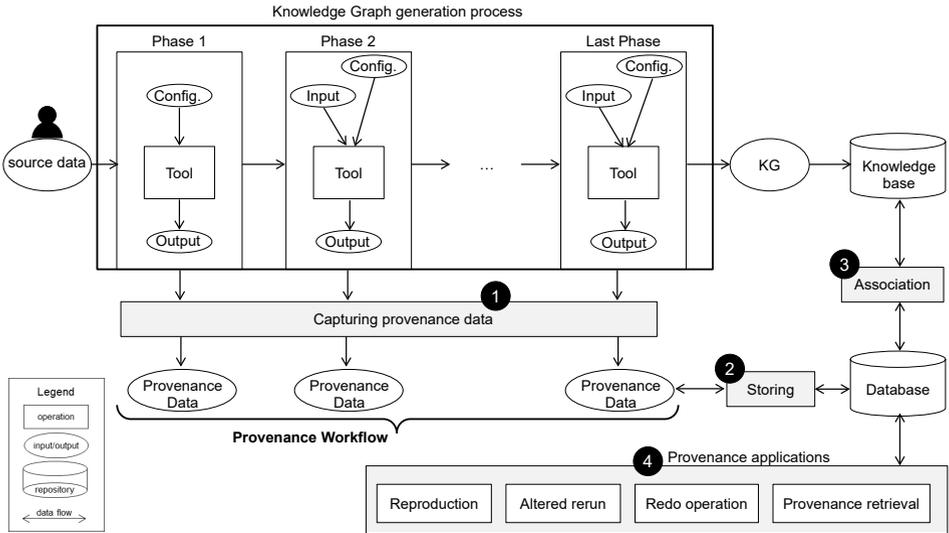


Fig. 1: Overview of our management framework for managing the provenance of the Knowledge Graph generation in a web portal.

## 2 Literature Review & Preliminaries

The provenance of an object is the history of its origin and derivation [MMW13]. Provenance tracking records the provenance of an object. In the literature, there have been different surveys (cf. [PRSA18, HDBL17]) on provenance characteristics and provenance models. The importance of provenance on large-scale KGs and the Web of Data has been highlighted in [Ho20]. As a solution to manage the provenance, computational Notebooks (cf. Jupyter Notebook [K116]) have gained widespread adoption in recent years, cf. ProvBook [SKR18]. However, implementing large, complex projects in a notebook, especially when multiple programming languages are used, is not straightforward. Another issue is the automation of a notebook. We faced different problems to connect a piece of software implemented in Jupyter Notebook to the backend of our studied platform, and executing cells when we receive specific user requests. On the other hand, web-based interactive development environments such as JupyterLab that can be hosted and accessed by multiple people would introduce security issues. Existing tools such as Open Refine (<https://openrefine.org/>) among others, can also track applied operations on the data and thus can be used as a provenance solution. However, we did not use such tools as a provenance solution, as they are not a fully-fledged development environment and it is not always possible to extend those tools with arbitrary code and still make use of its provenance features. To the best of our knowledge, a few KG platforms [Sy22, SDA20, Ha19, Be20] apply a provenance solution in some capacity. Of those, Blue Brain Nexus [Sy22] is the only one explicitly mentioning the importance of provenance data and their usage of the W3C PROV ontology [Le13].

The other platforms did not explain their approach on provenance management in their publications.

**Preliminaries.** To generate a KG, different phases need to be carried out (see Figure 1, top), where in each phase a (different) tool will be executed. Definition 1 shows our definition of the KG generation process.

**Definition 1** *KG generation is an ordered execution of tools in different phases, where source data  $d_s$  is the input and a Knowledge Graph (KG) is the output. Formally, this yields  $KG \leftarrow \text{generation}(d_s)$ .*

The tools are executed to perform different computational tasks e.g., cleaning datasets, linking the entities to external resources, and generating specific output. In this paper, we define a tool as:

**Definition 2** *A tool (TO) is an executable piece of software that performs some computational task. The input of a tool is a file along with a configuration. The output of a tool is a new processed file.*

A configuration is a set of input parameters for executing the tool. Every configuration is a set of key-value pairs, containing the name of a parameter and its value. For simplicity, we assume that tools are run sequentially to generate a KG. This execution order needs to be preserved. We assume the input of a tool execution is the output of the prior tool execution. Every tool execution has a file as an input and a new file as an output. We define any such file as a data object.

### 3 Our Proposed Provenance Management Framework

Figure 1 shows our provenance management framework in the KG generation of a web portal. The users go through different phases to generate the KG based on their source data. The resulting KG is added to the knowledge base. Thus, with each generation of a new (sub-)KG, the overall KG (in the remainder of the paper referred to as main KG) is extended. The provenance data of each phase is captured (❶), stored (❷), and associated with the generated KG (❸). The provenance data can be used for various applications (❹).

#### 3.1 Capturing provenance data

The provenance capture mechanisms collect information related to the KG generation process. In the web portal, each KG generation starts by uploading source data  $d_s$  by a user.

After that, the user selects a tool with a specific configuration and then the tool gets executed. This happens sequentially multiple times until the KG is generated. In our framework, we capture (see ❶ in Figure 1) the provenance of each data object, that got processed by a tool execution during the KG generation process. All executed tools, configurations, and input and output of each phase of the KG generation are saved separately. For each, we save a set of information such as the version of the tool or the storage location of a file. We capture all provenance data of an executed tool at every phase. We call the information about all executed tools in the sequential phases of the KG generation a workflow (see Definition 3). Note that, the provenance data of a data object includes all stored data of prior phases until that phase. In this view, the provenance data of the generated KG is the complete workflow  $W$ .

**Definition 3** A *workflow*  $W$  is an ordered collection of provenance data of executed tools in all phases of the KG generation process.

### 3.2 Provenance storage and association

During KG generation, we store (see ❷ in Figure 1) the provenance of each produced data object. Note that, each generated KG is a sub-KG of the main KG in the web portal. We consider two approaches for provenance storage and associating provenance data to a generated sub-KG (see ❸ in Figure 1).

**Approach 1:** Provenance data is saved in the knowledge base. This would require the provenance data to be represented as triples. These triples could be grouped by another graph name. The provenance data can then be coupled by a triple featuring both the provenance data of graph name and the KG.

**Approach 2:** The generated KGs are stored in a knowledge base, while their provenance data are stored in a relational database. To handle the provenance storage in the web portal, we store provenance data of KG generation process (i.e., workflow) as a JSON-string inside the provenance record. Listing 1 shows a layout of a workflow  $W$  holding for  $Phase_i$ . For every new phase, a new key  $Phase_{i+1}$  gets created together with a new dictionary and new values.

```
{ "Phase_i":  
  {  
    "input": "referenceTo(data object)",  
    "tool": "referenceTo(T0)",  
    "configuration":  
    {  
      "argname1": "value1",  
      "argname2": "value2",  
      ...  
    },  
    "output": "referenceTo(data object)"  
  }  
}
```

List. 1: Example of a workflow holding one phase in JSON-string.

For each request of KG generation in the web portal, we create a new entry in the relational database (provenance information of KG generation's phases). Table 1 shows a simplified version of this database (the database, including provenance data of tools, datasets, and users, are not shown here). Each KG generation process has a primary key and belongs to a specific project. All provenance data of a KG generation's phases (i.e., the workflow) is saved as a provenance record. We support here both graph- and triple-levels.

At the graph-level, we store the reference (URI of the sub-KG in the knowledge base) to this sub-KG in our relational database (Column 3 of Table 1). In this case, for each saved sub-KG in the knowledge base, we save the reference (id) of that sub-KG next to its provenance data. In the triple-level, subjects, predicates, and objects of all triples are annotated (e.g., via `rdfs:seeAlso`, `rdfs:comment`, or other defined annotations) with their respective provenance ids. With the provenance id we mean the first column of Table 1. If a user wants to retrieve the provenance of a specific sub-KG or a term (subject, predicate, or object), we can lookup for its provenance data via the reference of the sub-KG. The association is of type "no-coupling" according to [PRSA18]. The associated provenance data can be stored in different formats such as JSON, XML, or turtle (an RDF KG relying on the PROV-O ontology (see next section)). In this approach, provenance data is stored in the relational database and can be exported in different formats.

## 4 Implementation

We have partially tested our provenance management framework in the iKNOW project (<https://planthub.idiv.de/iknow/>), which is distributed under an open-source license in <https://github.com/fusion-jena/iKNOW>. iKNOW [Ba21] is a planned semantic-based toolbox for Knowledge Graph creation and evolution in the biodiversity domain. For the

Tab. 1: Provenance data about the KG generation process in the web portal.

| Key | Project Name | RefTo Sub-KG | Provenance Record  |
|-----|--------------|--------------|--|
| 001 | NameA        | $ex : G_1$   | [[("phase","1"), (input,"file12"), (tool,"ToolA"), (output,"file33"),(config,"arg1")]<br>[("phase","2"), (input,"file33"), (tool,"ToolD"), (output,"file53"),(config,"arg3")]<br>[("phase","3"), (input,"file53"), (tool,"ToolB"), (output,"file22"),(config,"arg6")]]   |
| 002 | NameB        | $ex : G_2$   | [[("phase","1"), (input,"file10"), (tool,"ToolC"), (output,"file44"),(config,"arg1")]<br>[("phase","2"), (input,"file44"), (tool,"ToolE"), (output,"file42"),(config,"arg4")]<br>[("phase","3"), (input,"file42"), (tool,"ToolG"), (output,"file12"),(config,"arg5")]<br>[("phase","4"), (input,"file12"), (tool,"ToolF"), (output,"file68"),(config,"arg8")]] |

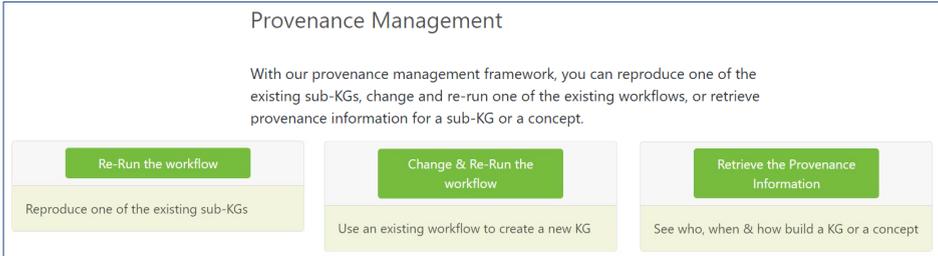


Fig. 2: Provenance Management GUI.

backend, we used the Python web framework Django ([www.djangoproject.com/](http://www.djangoproject.com/)). For building user interfaces on the frontend, we used Svelte (<https://svelte.dev/>). We used Docker ([www.docker.com/](http://www.docker.com/)) to encapsulate the different pieces of software that are - or will be - implemented on our server. A Docker container packages up code and all associated dependencies. This prevents dependency issues and provides an isolated runtime environment that can be used to serve all kinds of tasks. We used PostgreSQL ([www.postgresql.org/](http://www.postgresql.org/)) for saving data of the portal functionalities along with provenance data, and Blazegraph DB ([www.blazegraph.com/](http://www.blazegraph.com/)) for storing and accessing KGs. We currently implemented the second approach of provenance association.

## 5 Provenance Applications

Figure 2 shows the GUI of provenance management. Our provenance framework can support four different applications (see ④ in Figure 1):

**1. Reproducibility.** A reproducible KG can increase trust in the information provided and support open science. Reproducibility of a KG is the capability of getting the same KG by recreating or reproducing the KG. Reproducing the KG in our framework can be done by automatically running a pre-existing workflow of a sub-KG with the goal of reproducing the same KG. Through our GUI, the user can select one of the existing workflows and run the whole process. After executing the workflow, the resulting KG can be downloaded

separately. We also show the original version of the KG, so that the user can compare their triples and metadata (such as the number of triples).

**2. Altered rerun.** Let us consider this scenario, where a user wants to generate a new KG based on one of the existing workflows  $W$  of a pre-generated KG while possibly changing tools, configurations, or even the source data. To achieve this, we let the user select one of the existing workflows (see Figure 3), make the desired changes on that (such as selecting other tools or changing the tool's configuration), and then run the workflow over the same or another source data. The main advantage here is the possibility of generating a new KG automatically with an already known workflow  $W$ . This provides user convenience. In the end, we save this altered workflow as a new workflow in our portal.

**3. Undo operation.** Let us consider a user is in the process of generating a new sub-KG wanting to undo one or multiple tool executions. In this scenario, we let the user roll back one or several executed tool(s). Upon this action, the provenance data will be updated accordingly. Some additional implementation details (e.g., deleting files, or making provenance data and files consistent in the database) have to be considered to ensure the safety of the operation. This application is implemented in the Knowledge Graph generation scenario (see Figure 5).

**4. Provenance retrieval.** Retrieving provenance data is important for a user to view the data and understand how tools were executed during KG generation. Through our GUI (see Figure 4, top) users can select which sub-KG they want to retrieve provenance data. They can also retrieve the provenance data of a specific term. The system first search on which sub-KG the term exists. It then shows the list of sub-KGs to the user. Then, the user can select one of the sub-KG to see its provenance data. Figure 4, down, shows the result of provenance retrieval. The user can observe who, when and how the sub-KG is built. We currently offer the possibility to download provenance data of a sub-KG as a JSON. Moreover, our provenance data is mapped to the popular, standardized PROV-Ontology and can be downloaded in any RDF Syntax. We plan to provide downloading provenance data as a Turtle file, mapped to PROV-Ontology. A workflow can be mapped to the PROV-Ontology, considering the following rules:

- Every provenance data of each phase of KG generation gets a URI and becomes a `prov:Activity`.
- Every data object and configuration of a tool gets a URI according to its reference and becomes a `prov:Entity`.
- Every tool gets a URI according to its reference and becomes a `prov:Agent`.
- For every phase (provenance data of each phase), there is a triple with the URI of the phase as the subject, `prov:wasAssociatedWith` as the predicate and the URI of the tool as the object
- For every phase, there is a triple with the URI of the phase as the subject, `prov:used` as the predicate and the URI of the input data object as the object

### Choose one of collections

Show 10 entries Search:

| Collectionname  | Bioprojectname | # associated graphs | More                   |
|-----------------|----------------|---------------------|------------------------|
| sgpc_64_phenobs | phenobs        | 1                   | <a href="#">Choose</a> |
| sgpc_65_phenobs | phenobs        | 1                   | <a href="#">Choose</a> |
| sgpc_70_phenobs | phenobs        | 1                   | <a href="#">Choose</a> |
| sgpc_77_test1   | test1          | 1                   | <a href="#">Choose</a> |
| sgpc_91_phenobs | phenobs        | 1                   | <a href="#">Choose</a> |
| sgpc_92_phenobs | phenobs        | 1                   | <a href="#">Choose</a> |

Showing 61 to 66 of 66 entries Previous 1 2 3 4 5 6 7 Next

[Clear Choice](#)

### Chosen collection:

| Collection Name | Bioproject Name | # associated graphs | PK |
|-----------------|-----------------|---------------------|----|
| sgpc_92_phenobs | phenobs         | 1                   | 92 |

[Change & Re-Run](#)

Workflow Only
Workflow & dataset(s)

Your Dataset is: original.csv [Change dataset](#)

| Phase Name            | Method       | System generated result | User Changes | Final Result |
|-----------------------|--------------|-------------------------|--------------|--------------|
| Column Type Selection | iknow-method | Download                | Download     | Download     |
| Linking               | Direct API   | Download                | Download     | Download     |
| Property Declaration  | iknow-method | Download                | Download     | Download     |
| Schema Refinement     | iknow-method | Download                | Download     | Download     |
| Query Building        | iknow-method | Download                | Download     | Download     |
| Saving- Pushing       | iknow-method | Download                | Download     | Download     |
| Saving- Pushing       | iknow-method | Download                | Download     | Download     |

[Confirm & Execute](#)

Fig. 3: The GUI of altered rerun scenario.

### Provenance Retrieve

First select a sub-KG or a term that you would like to see its provenance information

Retrieving Provenance Information of

a sub-KG:  Show Provenance

a term:  Find Provenance

### Result of Provenance Retrieval

Here You see the provenance information of your selected sub-KG or term

**Who**  
Admin

**When**  
2022-12-01

**How**  
Here is the workflow of your selected sub-KG

Your Dataset is: original.csv

| Phase Name                         | Method       | System generated result | User Changes | Final Result |
|------------------------------------|--------------|-------------------------|--------------|--------------|
| Column Type Selection              | iknow-method | Download                | Download     | Download     |
| Cells Linking Property Declaration | Direct API   | Download                | Download     | Download     |
| Schema Refinement                  | iknow-method | Download                | Download     | Download     |
| undefined                          | iknow-method | Download                | Download     | Download     |
| undefined                          | iknow-method | Download                | Download     | Download     |

Download the workflow

Fig. 4: Top: Users select which term or sub-KG they want to retrieve the provenance; Down: It shows the result of provenance retrieval.

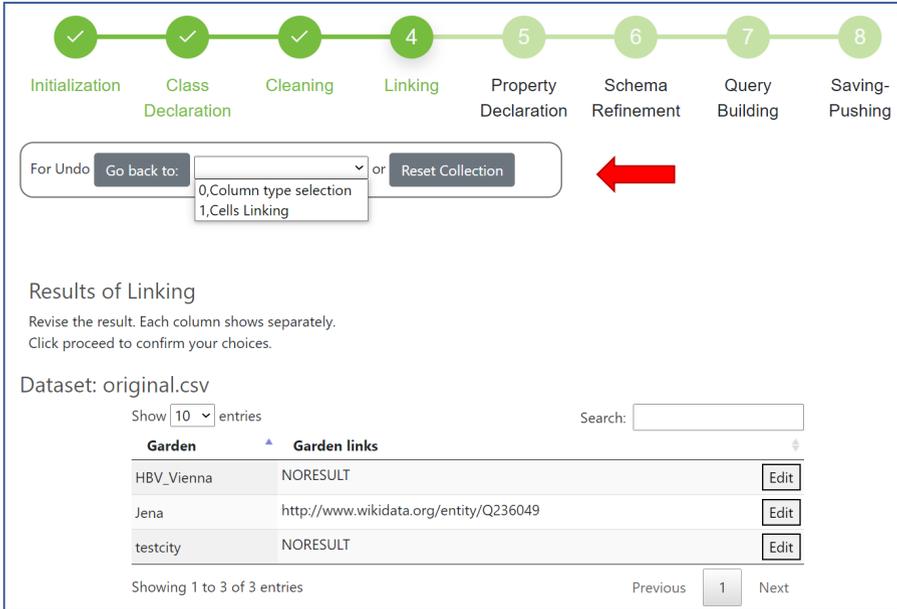


Fig. 5: Undo operation in the Knowledge Graph generation process.

- For every output data object there is a triple with the URI of the output data object as the subject, `prov:wasGeneratedBy` as the predicate and the URI of the according phase as the object
- The configuration gets a URI according to the index of the phase and becomes a `prov:Entity`. For every key-value pair in the configuration, a new triple can be created with the URI of the configuration as the subject.
- To show that the configuration is used in the phase, a triple `<ex:phase_i prov:used ex:config_i>` is generated.
- Additionally, the triple `<ex:output prov:wasDerivedFrom ex:input>` can be generated to show that the output data object was derived from the input data object.

Listing 2 shows an example of a phase in PROV format in the turtle syntax. The URIs of phase, tool, configuration, input, and output data objects are simplified to `ex:phase_i`, `ex:tool`, `ex:config_i`, `ex:input_i`, and `ex:output_i`, respectively. The URI of a tool `ex:tool` does not need an index, because we always have a finite set of known tools and no new tools get generated during tool execution. For every other subject, the URIs can be individualized e.g., by appending the index of phase `i`. For every following phase, the URI of the last output data object `ex:output_i` can be used without generating a new URI e.g.,

`ex:input_i+1`. In general, the URIs of data objects do not necessarily have to contain the terms `input` or `output`. This is just for explanatory purposes. It is only important that these URIs are unique.

```
ex:tool a prov:Agent .
ex:config_i a prov:Entity .
ex:phase_i a prov:Activity ;
    prov:used ex:input ;
    prov:wasAssociatedWith ex:tool ;
    prov:used ex:config_i .
ex:input_i a prov:Entity .
ex:output_i a prov:Entity ;
    prov:wasGeneratedBy ex:phase_i ;
    prov:wasDerivedFrom ex:input .
```

List. 2: Example of a phase in PROV format and the turtle syntax.

## 6 Conclusion & Future Work

In this paper, we provided the core concept and design of a framework to capture, store and retrieve provenance data of KG generation in a web portal and show an environment capable of provenance management. We presented four different applications to show the benefit of our proposed framework. However, the experimental test over this framework via a user study stays for our future work. Another future plan is extending the provenance capture and storage for tools with special requirements outside of our definition. This can involve e.g., multiple inputs and outputs of a tool. A possible solution to retrieve provenance data more efficiently and to enable retrieving triples based on the provenance data of the KG they belong to, can be achieved by saving provenance data in the same location as the KG is. In this way, a single query on the knowledge base can be issued, that filters results accordingly. Thus, we will handle this issue in our future work, too.

## Bibliography

- [Ba21] Babalou, Samira; Schellenberger Costa, David; Kattge, Jens; Römermann, Christine; König-Ries, Birgitta: Towards a Semantic Toolbox for Reproducible Knowledge Graph Generation in the Biodiversity Domain - How to Make the Most out of Biodiversity Data. In: INFORMATIK 2021. Gesellschaft für Informatik, Bonn, pp. 581–590, 2021.
- [Be20] Berven, Arne; Christensen, Ole A; Moldeklev, Sindre; Opdahl, Andreas L; Villanger, Kjetil J: A knowledge-graph platform for newsrooms. *Computers in Industry*, 123:103321, 2020.

- 
- [Ha19] Haase, Peter; Herzig, Daniel M; Kozlov, Artem; Nikolov, Andriy; Trame, Johannes: metaphactory: A platform for knowledge graph management. *Semantic Web*, 10(6):1109–1125, 2019.
- [HDBL17] Herschel, Melanie; Diestelkämper, Ralf; Ben Lahmar, Houssein: A survey on provenance: What for? What form? What from? *The VLDB Journal*, 26(6):881–906, 2017.
- [Ho20] Hogan, Aidan: Web of data. In: *The Web of Data*, pp. 15–57. Springer, 2020.
- [K116] Kluyver, Thomas; Ragan-Kelley, Benjamin et al.: Jupyter Notebooks—a publishing format for reproducible computational workflows. In: *In Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). volume 2016. IOS Press, pp. 87–90, 2016.
- [Le13] Lebo, Timothy; Sahoo, Satya; McGuinness, Deborah; Belhajjame, Khalid; Cheney, James; Corsar, David; Garijo, Daniel; Soiland-Reyes, Stian; Zednik, Stephan; Zhao, Jun: *Prov-o: The prov ontology*. 2013.
- [MMW13] Majumdar, Rupak; Meyer, Roland; Wang, Zilong: Provenance verification. In: *International Workshop on Reachability Problems*. Springer, pp. 21–22, 2013.
- [PRSA18] Pérez, Beatriz; Rubio, Julio; Sáenz-Adán, Carlos: A systematic review of provenance systems. *Knowledge and Information Systems*, 57(3):495–543, 2018.
- [SDA20] Staar, Peter W. J.; Dolfi, Michele; Auer, Christoph: Corpus processing service: A Knowledge Graph platform to perform deep data exploration on corpora. *Applied AI Letters*, 1(2):e20, 2020.
- [SKR18] Samuel, Sheeba; König-Ries, Birgitta: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In: *ISWC (P&D/Industry/BlueSky)*. 2018.
- [SKR22] Samuel, Sheeba; König-Ries, Birgitta: End-to-End provenance representation for the understandability and reproducibility of scientific experiments using a semantic approach. *Journal of Biomedical Semantics*, 13(1):1, December 2022.
- [Sy22] Sy, Mohameth François; Roman, Bogdan; Kerrien, Samuel; Mendez, Didac Montero; Genet, Henry; Wajerowicz, Wojciech; Dupont, Michaël; Lavriushev, Ian; Machon, Julien; Pirman, Kenneth et al.: Blue Brain Nexus: An open, secure, scalable system for knowledge graph management and data-driven science. *Semantic Web*, (Preprint):1–31, 2022.



# MLProvLab: Provenance Management for Data Science Notebooks

Dominik Kerzel,<sup>1</sup> Birgitta König-Ries,<sup>2</sup> Sheeba Samuel<sup>3</sup>

**Abstract:** Computational notebooks are a form of computational narrative fostering reproducibility. They provide an interactive computing environment where users can run and modify code and repeat the exploration, providing an iterative communication between data scientists and code. While the ability to execute notebooks non-linearly benefits data scientists for exploration, the drawback is that it is possible to lose control over the datasets, variables, and methods defined in the notebook and their dependencies. Thus, in this process of user interaction and exploration, there can be a loss of execution history information. To prevent this, a possibility is needed to maintain provenance information. Provenance plays a significant role in data science, especially in facilitating the reproducibility of results. To this end, we developed a provenance management tool to help data scientists track, capture, compare, and visualize provenance information in notebook code environments. We conducted an evaluation with data scientists, where participants were asked to find specific provenance information from the execution history of a machine learning Jupyter notebook. The results from the performance and user evaluation show promising aspects of provenance management features of the tool. The resulting system, MLProvLab, is available as an open-source extension for JupyterLab.

**Keywords:** Data Science; Information Extraction; Provenance; Jupyter Notebook; Reproducibility

## 1 Introduction

Data science and machine learning (ML) techniques significantly impact the scientific community in developing relevant and practical applications for society. With the rapid publication of results in the data science and ML field, it is increasingly important for scientists also to be able to reproduce and recreate results. Jupyter notebook [KR+16] is one of the adopted approaches researchers use to publish results of their data science and ML projects to enable reproducible computational research. The notebooks are computational narratives that data scientists widely use in multiple ways for scientific computing, exploration, tutorials, documentation, interactive manuals, publications, etc. This is possible because the notebooks encapsulate code and explanatory text, computational results, visualizations, etc., in a single document. In addition to being a stand-alone tool, it is also integrated with different data science platforms like Kaggle, Colab notebooks, etc. As a result of exploration

---

<sup>1</sup> Friedrich Schiller University Jena, Germany dominik.kerzel@uni-jena.de

<sup>2</sup> Heinz-Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University Jena, Germany  
Michael Stifel Center Jena birgitta.koenig-ries@uni-jena.de

<sup>3</sup> Heinz-Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University Jena, Germany  
Michael Stifel Center Jena sheeba.samuel@uni-jena.de

and constant changes done in a data science pipeline, it is essential to understand how the results are derived under which choices and assumptions of researchers. Provenance plays a significant role to record and reference the history of results. This, in turn, helps data scientists to enable reproducibility [SK21]. However, scientists do not have direct access to the history of their frequent experimentation in these notebooks. The rapid development of data science and ML methods and algorithms results in new releases of libraries and modules. As a result, running old notebooks without knowing the versions of libraries and modules can cause execution errors and incompatibility issues. Another significant barrier is the possibility of running notebooks non-linearly. It becomes difficult to reproduce and get the same or close-by results without understanding how each cell and the variables defined in them are dependent on each other.

To address these issues, in this paper, we present MLProvLab as an extension to JupyterLab<sup>4</sup>, providing provenance management for reproducibility. This tool provides significant benefits for data scientists to track, compare, manage, and visualize provenance information of their computational experiments written in Jupyter notebooks. We can track, at runtime, the datasets, variables, libraries, and functions used in the notebook and their dependencies between cells. This is also visualized as a provenance dependency graph with temporal information. We evaluated its efficiency and features through a performance test and a user evaluation with 15 participants using practical tasks. The study shows that these 15 data scientists using MLProvLab for the first time correctly answered an average of 82% of the tasks they were provided in a machine learning notebook which was totally new to them and consisted of 55 executions.

## 2 Background and Related Work

Though data scientists use multiple tools and software for their computational tasks, writing code using programming languages like Python and R is common in data science and ML. The open-source libraries like Scikit-Learn<sup>5</sup>, PyTorch<sup>6</sup>, Numpy<sup>7</sup>, etc., provide accessible and reusable tools for data analysis and are heavily used for data science, ML, and deep learning applications. Jupyter notebooks that support over 40 programming languages, including Python and R, are widely used by millions of scientists. This is clearly seen in the availability of millions of notebooks on GitHub. Hence, in this research, we focus on the provenance management of Jupyter notebooks, with specific attention on providing support for the reproducibility of data science workflows.

Tools for capturing provenance from scripts and programs at different levels of granularity have been actively developed [Da12; Mc15; Pi15]. The noWorkflow tool [Pi15] is one such tool that captures the definition, deployment, and execution provenance of Python scripts. With the current wide adoption of Jupyter notebooks [KR+16], research works have

---

<sup>4</sup> <https://jupyterlab.readthedocs.io>

<sup>5</sup> <https://scikit-learn.org>

<sup>6</sup> <https://pytorch.org/>

<sup>7</sup> <https://numpy.org/>

also focused on tracking provenance from computational notebooks [Ca17; He19; Ho14; KM18; KP17; Ma21; PGS18; Pi15; Sh23; SK18; Wa20; We19]. Prov-o-matic provides a provenance-tracking extension for older versions of IPython Notebooks, which saves the provenance traces to Linked Data file [Ho14]. Another approach to track provenance in computational notebooks is by integrating noWorkflow [Pi15]. As a result, the features provided by noWorkflow are available in the IPython notebooks. However, this approach allows a python script to be run from inside IPython notebooks capturing the provenance of scripts instead of notebooks. Systems like Verdant [Ke19] are more closely aligned with MLProvLab. Verdant helps data scientists examine the execution history and notebook events. However, we analyze the code and provide artifacts used in the code and the dependencies between the cells based on the artifacts.

Recent approaches have also developed custom Jupyter kernels to trace runtime user interactions and automatically manage the lineage of cell execution [KP17; Ma21]. In their approach, the tool is developed as a separate Jupyter kernel, allowing users to update all cells affected by a change in a cell. This is possible by adding unique and persistent identifiers to each cell and providing references to results in other cells. This is different from our approach as these approaches [KP17; Ma21] introduce changes to the kernel and requires its installation.

Recent works have also focused on the provenance and model management of data science and ML pipelines beyond computational notebooks. An overview of conceptual, data management, and engineering challenges in the ML model management is given in [Sc18]. One of the data management challenges concerning the provenance management of ML is automatically tracking and querying model metadata. Several tools have been developed as metadata capturing systems in recent years [Or20; Va16; Za18]. ModelDB [Va16] provides a feature to manage ML models with metadata logging of metrics, artifacts, tags, and user information. Some systems track detailed provenance data by depending on the users to understand their complex schema and integrate their code with the corresponding API provided by the system [Sc17]. These provenance-capturing systems generally require users to actively configure their code, e.g., by annotating functions, hyperparameters, and operations. Due to the extra time and effort required, users may omit to configure and annotate their code. Therefore, tools that automatically extract and manage metadata are preferable to systems that require human intervention.

It is essential to provide provenance management without changing the code environment for the user. It is also essential that such platforms provide metadata management to all their users, irrespective of their skills and experience in data science. JupyterLab is a great basis for such projects, as shown in other works [Ke19]. Hence, in this paper, we target the users of JupyterLab and allow automatic provenance extraction from data science notebooks.

### 3 MLProvLab

Kerzel et al. [KSK21] describe the use case, challenges, and design goals of our data science and ML provenance management tool to automatically expose the metadata. Based

on the design goals, we present MLProvLab, a provenance management tool to track, manage, compare, and visualize the provenance of data science notebooks. The tool is available as an open-source extension for JupyterLab<sup>8</sup>. MLProvLab is composed of two components. A Python backend component provides event listeners for user interactions, an Abstract Syntax Tree (AST) generator for analyzing the code, and a core messaging plugin to request information from the kernel and notebook panel. On the visualization side, MLProvLab provides a Javascript frontend component that captures user interactions, renders visualization, and generates provenance graph. The tool contains provenance capture, visualization, comparison, and export modules.

**Provenance Capture.** The provenance capture module of MLProvLab collects and stores the provenance of a user session triggered by the start of the kernel. We call the lifetime of a kernel an *epoch*. For every new kernel, the provenance of epochs is created and stored in the notebook metadata. The tool defines event listeners for different user actions like the execution, addition, and deletion of a cell. When a code cell is executed, the cell content is returned to the backend. The executed code is then analyzed using Abstract Syntax Tree (AST) and string pattern matching techniques to get data provenance. We capture information on the definition and usage of variables, functions, and classes. The import statements are also tracked to extract information on the libraries and modules used and their version information. Additional operations are performed to find data sources for ML provenance management using string matching. In summary, the MLProvLab tracks and manages every variable declared in the cell, the dependencies of variables that are not defined in the evaluated cell, used datasets and the corresponding variables, imported libraries, and modules, etc.

**Provenance Visualization.** For the provenance visualization module, the MLProvLab uses a provenance graph to visualize the provenance of the notebook, including the execution order of cells and the data dependencies between cells. The tool can be invoked using the ‘MLProvLab’ button in the notebook toolbar. Figure 1 shows the provenance visualization graph of a sample ML notebook. The data sources and execution provenance are shown in the graph. A node is created in the graph for every cell in the notebook. Edges show the dependencies between cells using variables or methods declared in a cell. The outgoing edge from a node indicates that a data source was defined and is used in the other corresponding node. The colors of the nodes and edges represent their status. Cells that are colored orange represent cells with data sources, and green represents cells with output. Cell half colored with orange and green show that the cell contains both datasets and output. Users can move the sliders at the bottom of the panel to see the history of the changes and runs performed by the user. The ‘Epoch’ slider provides the history of the execution of the Jupyter Notebook every time a new user session of the kernel is started. The ‘Execution’ slider depicts the execution history of the Jupyter Notebook every time an event on the notebook cell is registered. Correspondingly, the information for the execution environment, datasets, and libraries used are shown to the user for the selected execution. The tool also shows the number of user sessions, executions, and execution time. MLProvLab also provides a

---

<sup>8</sup> <https://github.com/fusion-jena/MLProvLab/>

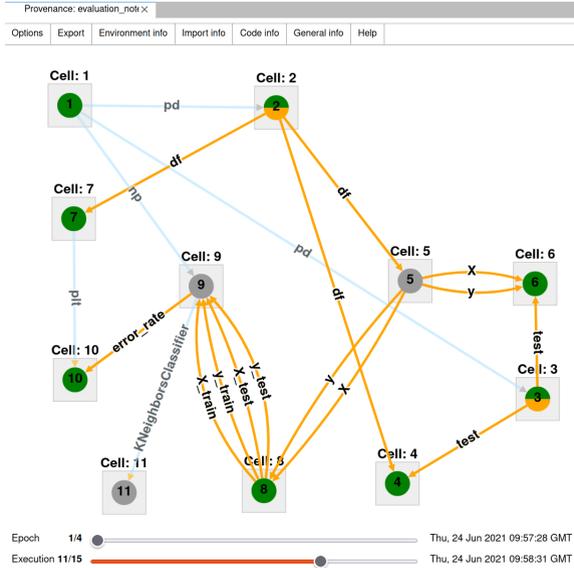


Fig. 1: Provenance Execution Graph in MLProvLab. The notebook cells are depicted as vertices of the graph. Green nodes in the graph show cells with any output type, orange show cells with data source or output is detected, grey shows cells where no data source or output is detected. Edges in the graph show the dependencies between cells. The footer allows to slide the execution history of the notebook.

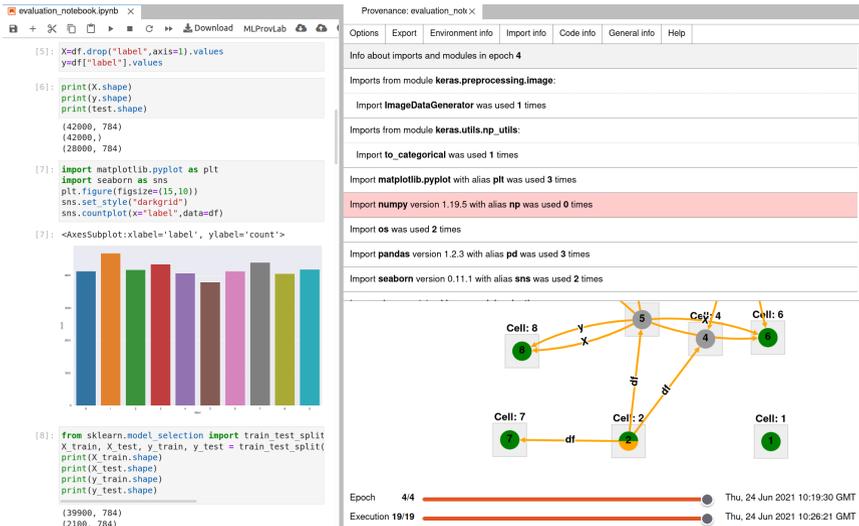


Fig. 2: Libraries and modules used in the notebook in the 4<sup>th</sup> epoch.

general menu with options to customize the graph to get additional provenance information. Figure 2 shows the *Import Info* menu in MLProvLab. It shows the information about the imported libraries, the module used in the notebook, and their version information. The libraries which are imported but not used are marked as red. To visualize the definition provenance, users can click on a node and open a radial context menu. This gives detailed information on the used datasets, functions, variables, outputs, etc. Users can also compare the definition provenance from previous runs. The graph is built using Cytoscape.js<sup>9</sup>.

| Provenance: evaluation_not x   |        |                  |             |           |              |      |
|--|--------|------------------|-------------|-----------|--------------|------|
| Options  | Export | Environment info | Import info | Code info | General info | Help |
| <b>Environment information of epoch 3</b>  |        |                  |             |           |              |      |
| Language: python   |        |                  |             |           |              |      |
| Version: 3.9.2   |        |                  |             |           |              |      |
| Mimetype: text/x-python  |        |                  |             |           |              |      |
| Kernel start time: Thu, 24 Jun 2021 10:17:35 GMT   |        |                  |             |           |              |      |
| Kernel implementation: ipython   |        |                  |             |           |              |      |
| Kernel version: 7.22.0   |        |                  |             |           |              |      |
| User agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0 |        |                  |             |           |              |      |

Fig. 3: Execution environment information of the notebook

| Provenance: evaluation_not x   |        |                  |             |           |              |      |
|--|--------|------------------|-------------|-----------|--------------|------|
| Options  | Export | Environment info | Import info | Code info | General info | Help |
| Source D:Projects\mnist-evaluation\data was first used in execution 1 in variable <code>df</code>            |        |                  |             |           |              |      |
| Source D:Projects\mnist-evaluation\data\train.csv was first used in execution 2 in variable <code>df</code>  |        |                  |             |           |              |      |
| Source D:Projects\mnist-evaluation\data\test.csv was first used in execution 3 in variable <code>test</code> |        |                  |             |           |              |      |

Fig. 4: Information on the datasets used in the notebook

Figure 3 and 4 shows the information of the execution environment of the notebook and the datasets used in the code, respectively. The execution environment of the notebook provides information on the programming language, kernel, operating system, and the versions of the selected epoch. Similarly, the *General info* provides information on the datasets used in each execution with their variable name.

**Provenance Comparison.** In the provenance comparison module of MLProvLab, the changes made to a notebook cell can be examined by users (Fig. 5). Users can select the execution of previous ML experiments and compare it with the current execution. We use the react-diff-view<sup>10</sup> component to visualize the differences.

**Provenance Export.** The provenance export module of MLProvLab allows users to export the collected provenance information of the notebook. Users can also clear the provenance history. However, users are given an alert to export the provenance before removing the provenance history from the notebook. This information is currently available in JSON format. For semantic interoperability, we plan to make this information available in other formats, including JSON-LD, RDF, etc.

<sup>9</sup> <https://cytoscape.org/>

<sup>10</sup> <https://github.com/otakustay/react-diff-view>

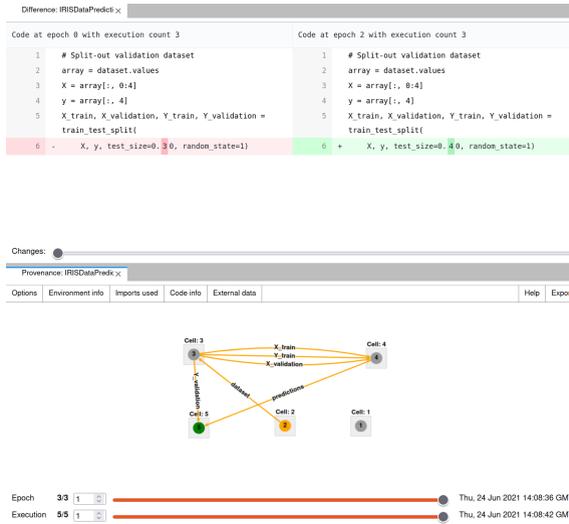


Fig. 5: Code difference between 2 different epochs

## 4 Evaluation of MLProvLab

The primary goal of our evaluation was to gather information about how the features of MLProvLab assist data scientists in the provenance management of Jupyter notebooks. As a result, we conducted a performance evaluation to test how MLProvLab handles notebooks with different numbers of cells. We also conducted a user evaluation to gather information on the impression of MLProvLab and test how the provenance management helps them understand the notebooks.

### 4.1 Performance Evaluation

We did a performance test with two different notebooks, one with 25 notebook code cells and the other with 100 code cells. Based on the study of analyzing 1 million computational notebooks on GitHub [RTH18], the typical number of total cells in a notebook range from 25-30. Notebook with more than 100 cells is infrequent. Hence, we selected notebooks with 25 and 100 cells. For the performance test, we used notebooks that calculate the Fibonacci number given an input. We defined ten variables, with each calculating a Fibonacci number for ten. These ten variables were assigned to each other and were repeated in the other cells. This was done to get multiple dependencies between every cell. As a result, each node in the provenance graph had ten outgoing edges to the other node.

Table 1 shows the status of the notebook run scenarios used for the evaluation. We evaluated six notebook run scenarios with different configurations. For example, in Notebook Run

Scenario A, the notebook with 100 cells is executed without enabling and updating the MLProvLab extension and provenance graph. While in Scenario B, the notebook has the same number of cells, with both MLProvLab extension and provenance graph enabled, and in Scenario C, MLProvLab enabled but without updating the provenance graph. These scenarios are also repeated with a notebook with 25 cells. We measured the execution time of the notebook in each scenario. Table 2 shows the results from the performance test for each notebook scenario. Each notebook is tested with different numbers of epochs (1, 2, 3, 4, 5, 10, and 20). The execution time in seconds for each notebook run scenario for each epoch count is shown.

| Notebook Run Scenario | Count of code cells | MLProvLab enabled | Provenance Graph Updated |
|-----------------------|---------------------|-------------------|--------------------------|
| A                     | 100                 | no                | no                       |
| B                     | 100                 | yes               | yes                      |
| C                     | 100                 | yes               | no                       |
| D                     | 25                  | no                | no                       |
| E                     | 25                  | yes               | yes                      |
| F                     | 25                  | yes               | no                       |

Tab. 1: Definition of the notebook run scenarios

| Epoch | Time in seconds |       |       |      |      |      |
|-------|-----------------|-------|-------|------|------|------|
|       | A               | B     | C     | D    | E    | F    |
| 1     | 2.46            | 89.06 | 4.13  | 1.02 | 5.83 | 1.66 |
| 2     | 2.39            | 90.94 | 5.58  | 1.05 | 6.64 | 1.75 |
| 3     | 2.50            |       | 7.37  | 1.02 | 6.24 | 1.54 |
| 4     | 2.47            |       | 8.56  | 1.01 | 6.21 | 1.99 |
| 5     | 2.48            |       | 10.05 | 1.02 | 6.37 | 2.21 |
| 10    |                 |       | 21.02 |      | 7.97 | 2.62 |
| 20    |                 |       | 24.18 |      | 8.11 | 3.85 |

Tab. 2: Performance evaluation of MLProvLab

| Notebook size | Epochs | Count of notebook code cells |
|---------------|--------|------------------------------|
| 12.0 KB       | 0      | 25                           |
| 2.5 MB        | 10     | 25                           |
| 5.0 MB        | 20     | 25                           |
| 36.0 KB       | 0      | 100                          |
| 9.9 MB        | 10     | 100                          |

Tab. 3: Cell count and size of evaluation notebooks

As seen in Table 2, the execution time of the notebooks increases as the number of epochs increases. For notebook scenario B, where there are 100 cells and both MLProvLab and the provenance graph are enabled and updated, we can see the overhead in loading the notebook. If one compares the runs with enabled and disabled extensions, one quickly sees

that a constant overhead is added. It is also noticeable that the execution times increase the more often the notebook is executed. This behavior is expected, as additional computations must be performed in the backend. Users working with notebooks with the average number of cells (around 25-30) are unaffected. However, working with notebooks with around 100 code cells with and without MLProvLab with multiple executions results in overhead in time. The overhead is due to the recomputation of the graph. However, due to this limitation, MLProvLab provides an option to disable the recalculation of the graph after every cell execution. This can further minimize the general overhead.

Table 3 shows the statistics based on the size of the notebooks and the number of execution. As expected, the size of the provenance information increases with the number of executions and code cell count. Currently, the captured provenance information is stored in the metadata of the notebook. This is located in a JSON object that has to be rewritten to disk after each update. Notebook containing the provenance information benefits users to share their intermediate and negative results, their choices and assumptions made during experimentation, etc. Currently, MLProvLab allows users to export the collected provenance data and then remove the provenance information if the notebook size gets too large. We plan to provide users an option to efficiently store the data, e.g., in SQLite database outside of JupyterLab.

## 4.2 User Evaluation

We present the materials and methods used and the results of the user evaluation conducted to get the impression of MLProvLab.

**Participants.** We used convenience sampling for the recruitment of participants. Participation in the user evaluation was voluntary. Forty participants responded to the survey, of which 36 agreed to the consent form and filled in their research background. However, only 15 participants finished the user evaluation and submitted their responses. We believe that this was due to the relative long time (around 25 minutes) needed to complete the tasks. Participants who read and agreed to the informed consent form and submitted their full responses were included in the final study. Of 15 participants, 14 have a computer science background, and 1 has physics. Seven undergraduate students, 2 Master students, 4 PhD Students, 1 PostDoc, and 1 Professor participated in the user evaluation.

**Materials.** We explored many publicly available data science notebooks to create a realistic provenance history for the experimentation. We also selected the evaluation notebook, which is not difficult for the participants to understand in minimum time. We used the Digit Recognizer problem from Kaggle, which uses the MNIST dataset<sup>11</sup>. We adapted the code and the resulting notebook contained 19 code cells with the provenance information collected from 4 epochs and 55 executions.

The questionnaire for the user evaluation was designed and developed using the following resources: (1) interviews conducted with the data scientists [SLK21] in the Werkstatt project

<sup>11</sup> <https://www.kaggle.com/c/digit-recognizer>

[Sa20] and (2) existing published literature on computational research reproducibility [Pi20]. The interviews and the existing literature provided insights into the challenges and problems faced by scientists and the provenance information required in reproducing published results of others in the context of data science and ML. The questionnaire was developed in English. A group of three researchers from computer science provided feedback on the length of the questionnaire, the priority and clarity of the defined questions, and technical issues in filling out the questionnaire. Based on the feedback, changes were made to the final version of the questionnaire.

The evaluation consisted of 26 questions grouped in 6 sections. The six sections are (1) Informed Consent Form (2) Research context of the participant (3) Testing MLProvLab with evaluation notebook (4) MLProvLab Introduction (5) Questions for Evaluation (6) General Impressions of MLProvLab. In the first section, we asked the consent from the respondents to participate in the evaluation. The informed consent form contained information about the study's background, purpose, procedure, voluntary participation, and contact information. Other than the informed consent form, none of the questions in the evaluation were mandatory.

In the second section, we asked about the research background of the participants. In addition to their current domain and position, we asked the participants whether they use Jupyter Notebooks and machine learning in their work. In the third section, we asked the participants to open the evaluation notebook and in the following section, we introduced MLProvLab. We provided a short tour showing its features and how they worked. This help page included screenshots and annotations of each feature provided by the tool. In the fifth section, we provided the questions to answer based on the evaluation notebook using MLProvLab. This section included 13 questions. Some of the questions were either single-choice or multiple-choice questions. Here is a list of the questions:

- Q1** Which version of the kernel was used in epoch '1'?
- Q2** Which external modules were used in epoch '1'?
- Q3** Are there any imported modules that were not used in epoch '3'?
- Q4** Which one was the most used module in epoch '3'?
- Q5** Which data sources were used in the notebook?
- Q6** In which execution and epoch the following figure got printed?
- Q7** Which version of seaborn was used?
- Q8** Which versions of python were used in the notebook?
- Q9** When was the notebook last executed?
- Q10** Are there any differences in the python and kernel versions used in the notebook in different executions?

- Q11** Which cells of the notebook in epoch ‘4’ are dependent on the variable ‘X\_train’?
- Q12** What is the accuracy score of the experiment in epoch ‘2’ when RandomForestClassifier was used?
- Q13** Has the train-test split ratio for the dataset changed during different executions?

In the last section, we asked the participants about their general impression of MLProvLab. In the first question, we asked how important is each MLProvLab module for the provenance management of computational experiments. We used a 5-point Likert scale for the answer options from *Very Easy* to *Very Difficult*. We also asked how easy it is to find provenance information on data science scripts using MLProvLab. In the next question, we asked the users to rate the perceived usefulness of MLProvLab. We asked whether they would like to use MLProvLab in their daily work. In the end, we provided an open-response question to participants to provide comments regarding the new features or changes they would like to see in MLProvLab. The average time taken to answer the evaluation questions in the notebook was 8 minutes. However, the average interview time, including the MLProvLab tour, was 28 min. The extra loading time of the Binder instance and the tour of MLProvLab could be some reasons for the long interview time.

The online evaluation was implemented using LimeSurvey<sup>12</sup>. The evaluation notebook is available in GitHub and was hosted using Binder<sup>13</sup>. Binder allows users to open the notebook with its execution environment, making the code and the extension (in this case, MLProvLab) available to everyone. We used a Jupyter notebook with Python version 3 to analyze the evaluation results. The source code and the results are available in GitHub repository<sup>14</sup>.

**Methods.** We sent invitations for participation to the PhD, Master, and Bachelor students in the Fusion group of the Computer Science Department of the University of Jena, Germany, and the Werkstatt project’s collaborating partners.

**Results.** Of 15 participants, 80% use Jupyter notebooks regularly or sometimes in their work. 66.67% of participants use Machine Learning regularly or sometimes in their work. Analyzing the results from the evaluation tasks of the notebooks, we see that 82% of answers to each question were correct, while 18% of answers were wrong. Three participants answered all the questions correctly. Eleven participants answered more than 60% of the questions correctly. However, the one participant who scored 46% did not attempt four questions and partially answered three correctly. For multiple-choice questions, we mark the answer correct only if the participants select all the right options. We observe that the multiple-choice questions were answered incorrectly, in particular for Question **Q11**, where seven participants gave the wrong answer. Questions **Q1** and **Q8** were answered correctly by every participants, followed by Questions **Q3**, **Q7**, **Q10** and **Q12**. For Question **Q11**, none of the persons selected the wrong option, but all the correct options were not marked.

<sup>12</sup> <https://www.limesurvey.org/>

<sup>13</sup> [https://mybinder.org/v2/gh/fusion-jena/MLProvLab/HEAD?urlpath=lab%2Ftree%2Fbinder%2Fevaluation\\_notebook.ipynb](https://mybinder.org/v2/gh/fusion-jena/MLProvLab/HEAD?urlpath=lab%2Ftree%2Fbinder%2Fevaluation_notebook.ipynb)

<sup>14</sup> <https://github.com/fusion-jena/MLProvLab>

The majority of the questions were answered correctly, which matches the impression given

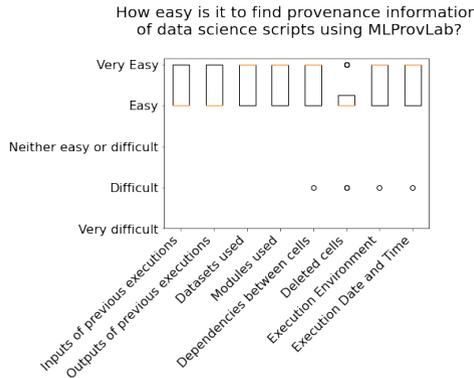


Fig. 6: Ease of finding provenance information of data science scripts using MLProvLab

by the users as shown in Figure 6. Finding the inputs and outputs of previous executions, the datasets and modules used, dependencies between cells, execution environment, and temporal aspects of notebook execution are either very easy or easy to find using MLProvLab (Figure 6). We did not provide any questions/tasks related to deleted cells; hence, we see that some participants were not aware of this feature of MLProvLab and chose the difficult option. Figure 7 shows the perceived usefulness and importance of MLProvLab modules.

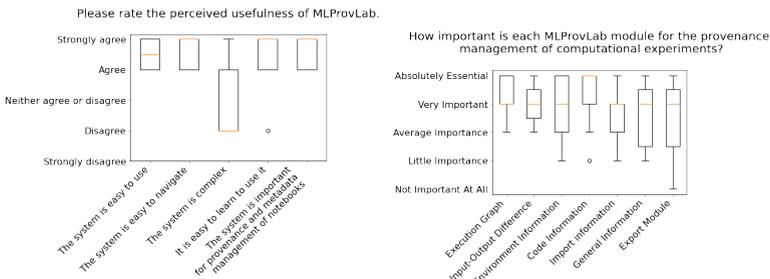


Fig. 7: Perceived usefulness and importance of MLProvLab and its modules for provenance management

Most of them marked that it was easy to use and navigate. Everyone agreed that the system is important for the provenance and metadata management of notebooks. Execution graph, Input-Output Difference, and Code Information are considered very important for users. The export information module was not considered important in the survey. We believe that this is because there were no tasks involving the export module. We received general comments from 11 participants for the open-response questions. The majority of them provided positive feedback. Some of the improvements provided by the participants include renaming the tabs to more meaningful names, rearranging tabs, creating a user option to directly input the epoch and execution number near the slider, and adding more general details in the Help tab as some features are not self-explanatory, and graph visualization not

following temporal order. Some suggested improvements are taken care of and implemented after the evaluation.

### **Limitations.**

This study was exploratory, and the sample needs to be more diverse to generalize the findings. Most of the participants have a computer science background. We expected more participation from other areas of study. Five participants from fields other than computer science did not complete the study. However, only 1 of these five respondents uses Jupyter Notebooks and Machine Learning in their work. This could be one of the reasons for not participating in the user evaluation. Our primary users are data scientists who use and have used Jupyter Notebooks. Most of the participants are students, but also other academic grades are represented. We also see many participants who do not use ML in their work. As a result, we also got opinions from users with and without experience in the domain. However, we have observed that each such participant has answered nine and more questions using MLProvLab.

## **5 Conclusions and Future Work**

We presented MLProvLab for the provenance management of data science notebooks. It is an extension of JupyterLab, to track, manage, compare, and visualize the provenance of notebooks. Through MLProvLab, users can efficiently and automatically track the provenance metadata, including datasets and modules used. We provide users the facility to compare different runs of computational experiments, thereby ensuring a way to help them make their decisions. The tool helps data scientists to collect more information on their experimentation and interact with them. It is designed so that the users do not need to change their scripts or configure them with additional annotations. In our future work, we aim to extend MLProvLab to identify the relationships between data and models for ML automatically. We want to track further the datasets and columns that have been used to derive the features of an ML model. This will help data scientists to get more information on the configurations used, e.g., hyperparameters, ML methods, etc. We also plan to provide interoperability by providing semantic annotations and descriptions of the collected fine-grained provenance information. We plan to use this provenance information to replay and rerun a notebook.

### **Acknowledgments**

The authors thank the Carl Zeiss Foundation for the financial support of the project “A Virtual Werkstatt for Digitization in the Sciences (K3)” within the scope of the program line “Breakthroughs: Exploring Intelligent Systems for Digitization - explore the basics, use applications” and Friedrich Schiller University Jena for the IMPULSE funding: IP 2020-10.

## References

- [Ca17] Carvalho, L. A. M. C.; Wang, R.; Gil, Y.; Garijo, D.: NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance. In: *SciKnow 2017*, Austin, Texas, 2017, 2017.
- [Da12] Davison, A.: Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. *Computing in Science Engineering* 14/4, pp. 48–56, 2012, ISSN: 1521-9615.
- [He19] Head, A.; Hohman, F.; Barik, T.; Drucker, S. M.; DeLine, R.: Managing Messes in Computational Notebooks. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019*, Glasgow, Scotland, UK, May 04-09, 2019. ACM, p. 270, 2019.
- [Ho14] Hoekstra, R.: PROV-O-Matic, <https://github.com/Data2Semantics/prov-o-matic>, Accessed 10 September 2021, 2014.
- [Ke19] Kery, M. B.; John, B. E.; O’Flaherty, P.; Horvath, A.; Myers, B. A.: Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, pp. 1–13, May 2019, ISBN: 978-1-4503-5970-2, visited on: 05/16/2021.
- [KM18] Kery, M. B.; Myers, B. A.: Interactions for Untangling Messy History in a Computational Notebook. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2018*, Lisbon, Portugal, October 1-4, 2018. IEEE Computer Society, pp. 147–155, 2018.
- [KP17] Koop, D.; Patel, J.: Dataflow Notebooks: Encoding and Tracking Dependencies of Cells. In: *9th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2017*, Seattle, WA, USA, June 23, 2017. USENIX Association, 2017.
- [KR+16] Kluyver, T.; Ragan-Kelley, B., et al.: Jupyter Notebooks-a publishing format for reproducible computational workflows. In: *ELPUB*. Pp. 87–90, 2016.
- [KSK21] Kerzel, D.; Samuel, S.; König-Ries, B.: Towards Tracking Provenance from Machine Learning Scripts. In: *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2021*, October 25-27, 2021. SCITEPRESS, 2021.
- [Ma21] Macke, S.; Parameswaran, A. G.; Gong, H.; Lee, D. J. L.; Xin, D.; Head, A.: Fine-Grained Lineage for Safer Notebook Interactions. *Proc. VLDB Endow.* 14/6, pp. 1093–1101, 2021, URL: <http://www.vldb.org/pvldb/vol14/p1093-macke.pdf>.
- [Mc15] McPhillips, T. et al.: YesWorkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. *arXiv preprint arXiv:1502.02403/*, 2015.

- [Or20] Ormenisan, A. A.; Ismail, M.; Haridi, S.; Dowling, J.: Implicit provenance for machine learning artifacts. *Proceedings of MLSys 20/*, 2020.
- [PGS18] Petricek, T.; Geddes, J.; Sutton, C.: Wrattler: Reproducible, live and polyglot notebooks. In (Herschel, M., ed.): *10th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2018*, London, UK, July 11-12, 2018. USENIX Association, 2018.
- [Pi15] Pimentel, J. F. N.; Braganholo, V.; Murta, L.; Freire, J.: Collecting and Analyzing Provenance on Interactive Notebooks: When IPython Meets No Workflow. In: *Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance. TaPP' 15*, USENIX Association, Edinburgh, Scotland, p. 10, 2015.
- [Pi20] Pineau, J.; Vincent-Lamarre, P.; Sinha, K.; Larivière, V.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; Larochelle, H.: Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *arXiv preprint arXiv:2003.12206/*, 2020.
- [RTH18] Rule, A.; Tabard, A.; Hollan, J. D.: Exploration and Explanation in Computational Notebooks. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. CHI '18*, ACM, Montreal QC, Canada, 32:1–32:12, 2018, ISBN: 978-1-4503-5620-6.
- [Sa20] Samuel, S.; Shadaydeh, M.; Böcker, S.; Brüggmann, B.; Bucher, S. F.; Deckert, V.; Denzler, J.; Dittrich, P.; von Eggeling, F.; Güllmar, D., et al.: A virtual “Werkstatt” for digitization in the sciences. *Research Ideas and Outcomes* 6/, 2020.
- [Sc17] Schelter, S.; Boese, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: *Machine Learning Systems Workshop at NIPS*. Pp. 27–29, 2017.
- [Sc18] Schelter, S.; Biessmann, F.; Januschowski, T.; Salinas, D.; Seufert, S.; Szarvas, G.: On Challenges in Machine Learning Model Management. *IEEE Data Eng. Bull.* 41/, pp. 5–15, 2018.
- [Sh23] Shankar, S.; Macke, S.; Chasins, S.; Head, A.; Parameswaran, A.: Bolt-on, Compact, and Rapid Program Slicing for Notebooks [Technical Report]./, 2023.
- [SK18] Samuel, S.; König-Ries, B.: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In: *International Semantic Web Conference (P&D/Industry/BlueSky)*. 2018, URL: <http://ceur-ws.org/Vol-2180/paper-57.pdf>.
- [SK21] Samuel, S.; König-Ries, B.: Understanding experiments and research practices for reproducibility: an exploratory study. *PeerJ* 9/, e11140, Apr. 2021, ISSN: 2167-8359, URL: <https://doi.org/10.7717/peerj.11140>.

- [SLK21] Samuel, S.; Löffler, F.; König-Ries, B.: Machine Learning Pipelines: Provenance, Reproducibility and FAIR Data Principles. In: Provenance and Annotation of Data and Processes - 8th and 9th International Provenance and Annotation Workshop, IPAW 2020 + IPAW 2021, Virtual Event, July 19-22, 2021, Proceedings. Vol. 12839. Lecture Notes in Computer Science, Springer, pp. 226–230, 2021.
- [Va16] Vartak, M.; Subramanyam, H.; Lee, W.-E.; Viswanathan, S.; Husnoo, S.; Madden, S.; Zaharia, M.: ModelDB: a system for machine learning model management. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics. Pp. 1–3, 2016.
- [Wa20] Wang, J.; Kuo, T.; Li, L.; Zeller, A.: Assessing and Restoring Reproducibility of Jupyter Notebooks. In: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020. IEEE, pp. 138–149, 2020.
- [We19] Wenskovitch, J.; Zhao, J.; Carter, S.; Cooper, M.; North, C.: Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure. In: 2019 IEEE Visualization in Data Science (VDS). IEEE, pp. 1–10, 2019.
- [Za18] Zaharia, M. et al.: Accelerating the Machine Learning Lifecycle with MLflow. IEEE Data Eng. Bull. 41/4, pp. 39–45, 2018, URL: <http://sites.computer.org/debull/A18dec/p39.pdf>.

# Data Extraction for Associative Classification using Mined Rules in Pediatric Intensive Care Data

Pronaya Prosun Das<sup>1</sup>, Marcel Mast<sup>2</sup>, Lena Wiese<sup>1,3</sup>, Thomas Jack<sup>4</sup>, Antje Wulff<sup>2,5</sup>, ELISE STUDY GROUP<sup>6</sup>

**Abstract:** Based on the characteristics of health and medical informatics, data mining techniques that were designed to tackle healthcare problems are faced with new challenges. One such challenge is to prepare medical data for pattern mining or machine learning. In this paper, we present a feature engineering technique for the Associative Classification of the Systemic Inflammatory Response Syndrome (SIRS) in severely ill children by mining Associative Rules. SIRS is characterized as the body's excessive defense response due to malevolent stressors such as trauma, acute inflammation, infection, malignancy, and surgery. It can have an impact on the clinical outcome and elevate vulnerability for organ dysfunctions. We aim to extract the features from given datasets using the described extraction process. After the transformation, those features are used to mine rules using Association Rule Mining. Those rules are used to perform Associative Classification and evaluated with the result generated by SIRS criteria defined by the experienced clinicians. The mined rules provide better control over sensitivity and specificity than the SIRS criteria used in everyday medical practice..

**Keywords:** Data Mining; SIRS; Association Rule Mining; Associative Classification

<sup>1</sup> Fraunhofer Institute for Toxicology and Experimental Medicine, Hannover, Germany. pronaya.prosun.das@item.fraunhofer.de

<sup>2</sup> Peter L. Reichertz Institute for Medical Informatics of TU Braunschweig and Hannover Medical School, Hannover, Germany.

<sup>3</sup> Institute of Computer Science, Goethe University Frankfurt, Frankfurt a. M., Germany.

<sup>4</sup> Department of Pediatric Cardiology and Intensive Care Medicine, Hannover Medical School, Hannover, Germany.

<sup>5</sup> Big Data in Medicine, Department of Health Services Research, School of Medicine and Health Sciences, Carl von Ossietzky University Oldenburg, Oldenburg, Germany.

<sup>6</sup> ELISE STUDY GROUP: Louisa Bode <sup>a</sup>; Marcel Mast <sup>a</sup>; Antje Wulff <sup>a, d</sup>; Michael Marschollek <sup>a</sup>; Sven Schamer <sup>b</sup>; Henning Rathert <sup>b</sup>; Thomas Jack <sup>b</sup>; Philipp Beerbaum <sup>b</sup>; Nicole Rübsamen <sup>c</sup>; Julia Böhnke <sup>c</sup>; André Karch <sup>c</sup>; Pronaya Prosun Das <sup>e</sup>; Lena Wiese <sup>e</sup>; Christian Groszweski-Anders <sup>f</sup>; Andreas Haller <sup>f</sup>; Torsten Frank <sup>f</sup>

<sup>a</sup>Peter L. Reichertz Institute for Medical Informatics of TU Braunschweig and Hannover Medical School, Hannover, Germany.

<sup>b</sup>Department of Pediatric Cardiology and Intensive Care Medicine, Hannover Medical School, Hannover, Germany.

<sup>c</sup>Institute of Epidemiology and Social Medicine, University of Muenster, Muenster, Germany.

<sup>d</sup>Big Data in Medicine, Department of Health Services Research, School of Medicine and Health Sciences, Carl von Ossietzky University Oldenburg, Oldenburg, Germany.

<sup>e</sup>Research Group Bioinformatics, Fraunhofer Institute for Toxicology and Experimental Medicine, Hannover, Germany.

<sup>f</sup>medisite GmbH, Hannover, Germany.

## 1 Introduction

Recently, patient monitoring and clinical documentation in the Intensive care unit (ICU) are performed with patient data management systems (PDMS). Sophisticated PDMS are able to directly present and analyze all incoming data, necessary for the diagnosis of a disease. Systemic inflammatory response syndrome (SIRS) was introduced along with the definition of sepsis for the adult in 1992 [Bo92] and SIRS criteria were modified for the children with age-specific norms in 2005 [Go05]. Four criteria were proposed to detect the presence of SIRS in children, which are a) hyperthermia or hypothermia, b) tachy- or bradycardia c) tachy- or bradypnoea or d) leukocytosis, leukopenia or increased immature neutrophile count [Go05]. Two of these four criteria must be evident, one of which has to be abnormal leukocyte count or temperature to be diagnosed with SIRS. For pediatricians, the recognition of SIRS in children is a challenging duty due to the nature of its definition, as there are different age-specific values for all these criteria except temperature. Age groups for some patients, especially newborn children, can even change multiple times within a short timespan i.e., during their stay on the pediatric intensive care unit (PICU) [In09]. Due to this complexity, it is difficult to diagnose SIRS in an early manner, especially in the stressful surroundings of a PICU. Several studies showed that any delay in recognizing SIRS and sepsis could increase mortality and morbidity significantly [Ha03]. Yet, by early identification of SIRS and sepsis, it is possible to prevent organ dysfunction and lead to a much better outcome for ICU patients.

In this work, a data mining technique is introduced to extract and transform different vital signs and laboratory tests from the ‘cross-institutional and data-driven decision-support for intensive care environments’ (CADDIE-2) dataset [Wu19] and to find associations between them. In clinical settings, these association rule mining approaches can be considered as supporting methods to better comprehend the disease patterns for the patients. Mainly, the goal of this work is four-fold: i) to explore a data extraction and transformation technique for SIRS, ii) to evaluate the extraction process using expert rules (SIRS criteria), iii) to get important features and iv) to find association rules for associative classification and evaluate it using the result produced by expert rules.

The rest of this article is organized as follows. Section 2 discusses prior work. We describe our feature engineering and extraction techniques in Section 3. In Section 4, we present the definition of expert rules and association rule mining (ARM). Results and discussion are provided in Section 5. Finally, we derive conclusions of the study in Section 6.

## 2 Prior Works

Hospital data are still, to a large extent, under-explored, despite growing awareness of their particular potential value in health analytics and risk modeling [De06, Ke13]. The variety and complexity of patient records present a substantial challenge for knowledge discovery. Generally, disease-specific data are gathered by various medical expertise; for instance, suicide risk assessments use a distinct data format than white blood cell counts.

It is apparent that hand-picking features for each analysis are inadequate, and it is also impossible to ensure that all substantial information in the data is incorporated. Nowadays, the use of Machine Learning (ML) approaches are escalating for knowledge discovery and prediction of biomedical data [Ta07, XJ19]. ARM is one of the areas of ML that can also be used for finding patterns in biomedical data; this commonly used data mining application determines the patterns of items or events [So17, Iv15]. To mine association rules for pattern discovery, several incremental techniques were presented recently [Aq19, LNFV21]. It was also utilized to solve various problems in the healthcare sector. Mainly, there are a number of links between patients' diseases and vital signs or symptoms. ARM can assist researchers to comprehend a disease effectively by finding those links. One study identified early childhood caries using ARM [Iv15]. Other authors [RN20] used ARM in conjunction with a keyword-based clustering strategy for the prediction of the disease. Risk factors of heart diseases were determined using ARM in some studies [Na13, So17]; while others [No18] identified the negative incidents caused by drug-drug interactions. The risk of diabetes mellitus was predicted using ARM in [Ka16]. Moreover, Borah et al. [BN18] identified different risk factors of breast cancer, hepatitis, and cardiovascular disease by applying dynamic rare ARM. However, the use of ARM to find a pattern for SIRS is absent in the literature which also highlights the importance of this work.

### 3 Data Preparation

This section provides an overview of the used dataset, extraction and transformation of the features for ARM. The overall procedure is illustrated in Figure 1.

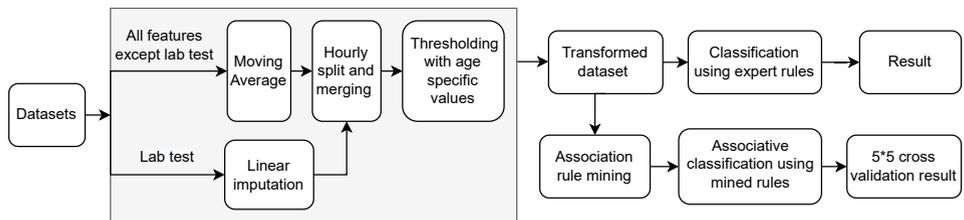


Fig. 1: Block diagram for the overall procedure.

#### 3.1 Initial Dataset

In the context of this work, we use routine data from the PICU of the Hannover Medical School [Wu21]. The data is not currently available for public use, but within the ELISE project we are creating an Evolutionary Open Pediatric Intensive Care Dataset [Rü22]. The dataset consists of various vital parameters like temperature values, heart and respiration rates as well as laboratory test results and information from medical devices such as cooling blankets, ventilation and pacemaker for each of the included 168 pediatric patients. The

patients can be distinguished from one another by a unique identifier (study number) that was generated as part of the pseudonymization. Laboratory test results include leukocyte, platelet and neutrophil counts as well as INR values derived from the prothrombin time. All measurements in the dataset come with a timestamp documenting its time of measurement by which a temporal sequence is ensured. Furthermore, the age of the respective patients is given. These parameters are supplemented by blood pressure values. In addition to the existing data, there is also a gold standard for the existence of SIRS for the respective patients for the period of the documented data available [Wu19, Wu21]. For the generation of this gold standard, two experienced pediatric intensive care physicians assessed the patients according to SIRS diagnostic rules defined by the International Pediatric Sepsis Consensus Conference (IPSCC) [Go05]. The clinician's decision on the presence of SIRS has been documented for each day that a particular patient was stationed at the PICU. In addition to this day-based gold standard, SIRS episodes were precisely documented in terms of time in order to provide an additional episode-based gold standard. The data set includes 168 patients from the pediatric intensive care in 243 days corresponding to a total of 1,998 days of stay within the ward. From those 1.998 days of stay, 460 days were labeled as SIRS within the day-wise gold standard [Wu21]. According to the gold standard 101 out of 168 patients suffered from SIRS during their hospitalization corresponding to a proportion of approximately 60%. The distribution of the sex is 106 to 62 in favor of male patients.

## **3.2 Feature Extraction**

We work with five different vital signs named temperature, pulse rate, respiration rate and systolic and diastolic pressures, and one lab test result. We use these features due to the orientation along with the IPSCC criteria. Birthdate, disease diagnosis and gender information of the patient are also being taken for further analysis. All of these features are extracted from their respective datasets and merged into a single table based on timestamps for further processing. The extraction process is described in the following subsections.

### **3.2.1 Moving Average**

In this approach, we obtain the moving average (MA) for temperature, pulse, respiration, systolic and diastolic pressures data, and used these averages for further processing. However, we do not apply MA to laboratory test values (leukocyte counts), as very few observations are available for it. Sometimes, there are only one or two laboratory values per 24 hours. A MA is a type of finite impulse response filter which is normally used in technical analysis by constructing a sequence of averages of different subsets of the full dataset. There are different variations existing in the literature (e.g., simple [Hy11], exponential [Br57],

weighted [Hy11], etc.). In our work, we have used the standard version of moving average of order  $m$  can be written as [Hy11],

$$\hat{T}_t = \frac{1}{m} \sum_{i=-k}^k y_{t+i} \text{ where } m = 2k + 1 \quad (1)$$

MA eliminates randomness in the data, leaving behind a consistent trend-cycle component. The trend-cycle at time  $t$  is estimated by taking the average of the values of the time series within  $k$  periods (in minutes) of  $t$ . We try to find the optimal  $k$  and experiment with different values from  $k = 10$  to  $k = 30$ . Based on the experiment, we choose  $k = 15$ , as the result remains almost unchanged when  $k > 15$ . We do not apply the MA operation to the lab-test dataset due to the unavailability of sufficient data points. The effect of the moving average is shown in Figure 2. It also improves the overall classification result as shown in Section 5.

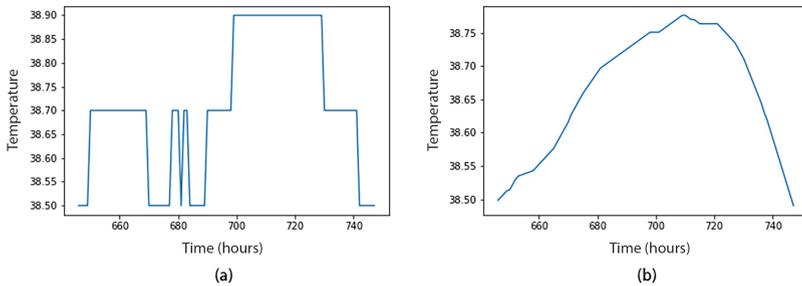


Fig. 2: The effect of moving average of the temperature data within a particular stay for a specific study number. (a) raw data. (b) after applying MA operation to the raw data of (a)

### 3.2.2 Hourly split

The features are split on an hourly basis after obtaining the moving average and merged into a single table. This produces missing values for laboratory tests due to a lack of hourly observation as mentioned previously. Table 1 shows the imputation for the feature ‘leukocyte count’. The column named Lab test represents the actual leukocyte measurement in each hour. The missing values are imputed before any further processing on this data as follows,

- The missing values between any two numerical observations are imputed by the equally spaced numbers of those two observations (linear imputation).

$$X_{\text{val}} = A_{\text{val}} + \frac{B_{\text{val}} - A_{\text{val}}}{\text{Number of observation between } A_{\text{val}} \text{ and } B_{\text{val}} \text{ (inclusive)} - 1} \quad (2)$$

where,  $X_{\text{val}}$  is the missing value,  $A_{\text{val}}$  is the actual assessment taken prior to  $X_{\text{val}}$ , and  $B_{\text{val}}$  is the the actual assessment taken after  $X_{\text{val}}$ .

- If a column starts with missing values, all the missing values prior a numerical observation are imputed with the same value of that observation.
- If a column ends with missing values, we look for the last observation with a numerical value and impute all the missing values after that with the same value of that observation.

Tab. 1: Imputation for lab test (leukocyte count). The actual lab test measurement and the imputed values are shown in Lab Test and Imputed columns, respectively.

| Study | Lab Test | Date-time From      | Date-time To        | Imputed |
|-------|----------|---------------------|---------------------|---------|
| 2     |          | 2018-08-01 16:00:00 | 2018-08-01 17:00:00 | 8       |
| 2     | 8        | 2018-08-01 17:00:00 | 2018-08-01 18:00:00 | 8       |
| 2     |          | 2018-08-01 18:00:00 | 2018-08-01 19:00:00 | 8.4     |
| 2     |          | 2018-08-01 23:00:00 | 2018-08-01 00:00:00 | 8.8     |
| 2     | 9.2      | 2018-08-02 00:00:00 | 2018-08-01 01:00:00 | 9.2     |
| 2     |          | 2018-08-02 01:00:00 | 2018-08-01 02:00:00 | 9.2     |

### 3.2.3 Thresholds for numerical values

Initially, we convert the age into age groups using Table 2. This table defined the High (H.), Low (L.) and Normal (N.) thresholds for the body vitals and laboratory test. Using these thresholds, we also convert the needed features (Temperature (Temp.), Pulse, Respiration (Resp.), Leukocytes (Leuk.), etc.) to categorical features.

## 3.3 Feature Transformation

One of our objectives is to apply expert rules to the extracted dataset that contains maximum, minimum, mean and median values for the vital signs and lab tests for the individual study numbers in a stay- and day-wise manner. We use the birth date from the Electronic Health Record (EHR) dataset and calculate the age in weeks for the specific study number by looking at the day/stay-based Gold standard datasets. Gender information is added as well from the Gender dataset. Feature Transformation transforms numerical values to categories using thresholds for numerical values.

As we have mentioned, five vital signs and one laboratory test are used in our experiment due to the fact that the SIRS criteria (expert rules / IPSCC criteria) only utilize those variables. Actually, blood pressure is not explicitly used in the four IPSCC criteria, although Goldstein et al. [Go05] listed norm-values for blood pressure. Therefore, we slightly modify the SIRS criteria by adding systolic and diastolic blood pressures to it. It increases the feature count as well as provides us with the impact of systolic and diastolic pressure on SIRS in rule mining. For each feature, four simple descriptive statistical summaries are

Tab. 2: Thresholds for age-specific vital signs and laboratory tests [Go05].

| Age Groups          | Pulse High (Beat/m) | Pulse Low (Beat/m) | Resp. (Breath/m) | Leuk. High | Leuk. Low | Sys. Normal | Dias. Normal | Temp. High | Temp. Low |
|---------------------|---------------------|--------------------|------------------|------------|-----------|-------------|--------------|------------|-----------|
| Newborn (0d-1wk)    | >180                | <100               | >50              | >34        | -         | 60-90       | 20-60        | >38.5      | <36       |
| Neonate (1wk-1m)    | >180                | <100               | >40              | >19.5      | <5        | 87-105      | 53-66        | >38.5      | <36       |
| Infant (1m-1y)      | >180                | <90                | >34              | >17.5      | <5        | 95-105      | 53-66        | >38.5      | <36       |
| Toddler (2-5y)      | >140                | <60                | >22              | >15.5      | <6        | 95-110      | 56-70        | >38.5      | <36       |
| School age (6-12y)  | >130                | <60                | >18              | >13.5      | <4.5      | 97-112      | 57-71        | >38.5      | <36       |
| Adolescent (13-18y) | >110                | <60                | >14              | >11        | <4.5      | 112-128     | 66-80        | >38.5      | <36       |

obtained (Maximum (Max), Minimum (Min), Median and Average (Avg)). Hence, only one summary at a time is used by the expert rules. For example, after transforming the maximum values of the Temperature (Temp.), Pulse, Respiration (Resp.), Systolic (Sys.) and Diastolic (Dias.) pressure, and Leukocytes (Leuk.) along with the age groups, the data looks like Table 3. Afterwards, the results (specifically accuracy, sensitivity and specificity) of expert rules are acquired for each summary.

Tab. 3: An example of a transformed dataset after thresholding.

| Pulse   | Resp. | Temp.  | Leuk.  | Sys.   | Dias.  | Age     | Diagnosis |
|---------|-------|--------|--------|--------|--------|---------|-----------|
| Tachyc. | High  | Normal | Normal | High   | Normal | Neonate | No SIRS   |
| Tachyc. | High  | High   | Normal | High   | Normal | Neonate | SIRS      |
| Normal  | High  | Normal | Normal | Normal | Normal | Newborn | No SIRS   |

## 4 Exploring Rules

Initially, the definition of SIRS criteria is presented. Then, frequency-based association rule mining along with their definitions is illustrated and associative classification is described.

### 4.1 SIRS criteria

The definition of SIRS is depicted here in simple terms. For diagnosing SIRS, at least two of the following four criteria must be present, one of which has to be abnormal leukocyte count or temperature [Go05]. (1) Pulse: high (tachycardia) or low (bradycardia); (2) Temperature: high or low (3) Respiration: high (4) Leukocyte count: high or low. We modify the expert rules (SIRS criteria) by adding a new rule for blood pressure: (5) Blood pressure: diastole high or diastole low and systole high or systole low. Here, high or low value means above or below age-specific norm values, respectively. The purpose of applying these expert rules is to understand the quality of the dataset and choose statistical features (maximum, minimum, average and median) for ARM.

### 4.2 Association Rule Mining

In ARM, association rules are extracted by mining transaction data to find out the relationships of different items inside the dataset. Assume, we have a transaction dataset with  $n$  transactions,  $T = \{t_1, t_2, t_3 \dots, t_n\}$  and  $m$  items,  $I = \{i_1, i_2, i_3 \dots, i_m\}$ . Here, each transaction is a set of items, therefore,  $t_c \in T$  with a distinct identifier TID is a subset of  $I$ . A transaction dataset is derived from the transformed dataset of Table 3 is shown in Table 4.

Tab. 4: A transaction dataset derived from Table 3.

| TID   | Transaction  |
|-------|--|
| $t_1$ | (Pulse tachyc.:1), (Resp. high:1), (Temp. normal:1), (Leuko normal:1),<br>(Syst. high:1), (Diast. Normal:1), (No SIRS:1) |
| $t_2$ | (Pulse tachyc.:1), (Resp. normal:1), (Temp. high:1), (Leuko high:1),<br>(Syst. high:1), (Diast. Normal:1), (SIRS:1)      |
| ..    |  |

Suppose,  $X$  and  $Y$  are the sets of items. Therefore, a rule is inferred by  $X \rightarrow Y$ . Here  $X$  and  $Y$  are known as antecedent and consequent, respectively. Also,  $X \cap Y = \emptyset$  and  $X, Y \subset I$ . We use support and confidence for rule mining.

**Support** It provides the notion of how frequent or popular an itemset is in all transactions.

$$\text{Support}(X \rightarrow Y) = \frac{\text{Number of transactions containing both } X \text{ and } Y}{\text{Total number of transactions}} \tag{3}$$

**Confidence** This metric specifies how regularly the association rule is identified to be authentic: for a given antecedent, it finds out the likelihood of occurrence of the consequent.

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Number of transactions containing both } X \text{ and } Y}{\text{Number of transactions containing } X} \tag{4}$$

Following algorithm is used for association rule mining:

```

Input :
min_conf : threshold value for minimum confidence;
itemset 1 : I1;
itemset 2 : I2;
Output :
Association Rule : I1 → I2;
X ← I1
Y ← I2
if X ⊆ Y then
    Confidence ← Support(X) ÷ Support(Y)
    if Confidence ≥ min_conf then
        | Association Rule ← I2 → I1
    end
end
Return Association Rule

```

**Algorithm 1:** ARM algorithm

### 4.3 Associative classification

By using a set of provided rules, specifically the class association rules, it is possible to design a rule-based classifier consisting of two phases [AT14]. Initially, the ARM technique is used to obtain a set of rules for the classifier. Afterwards, the rules are refined and joined together in order to construct the finalized rule-based classifier. Refining the rules requires pruning, tuning and ranking operations. The evaluation of the classifier can be performed on the test set using standard metrics. In this work, the accuracy, the sensitivity or true positive rate (TPR) and the specificity or true negative rate (TNR) are obtained for evaluation.

## 5 Results and Discussion

Tab. 5: Classification results of expert rules.

| Using MA | Summary | Accuracy | Sensitivity (TPR) | Specificity (TNR) |
|----------|---------|----------|-------------------|-------------------|
| Yes      | Maximum | 0.87     | 0.76              | 0.93              |
| Yes      | Median  | 0.86     | 0.76              | 0.93              |
| Yes      | Average | 0.86     | 0.76              | 0.93              |
| Yes      | Minimum | 0.86     | 0.76              | 0.92              |
| No       | Maximum | 0.86     | 0.76              | 0.92              |
| No       | Median  | 0.86     | 0.75              | 0.93              |
| No       | Average | 0.86     | 0.75              | 0.92              |
| No       | Minimum | 0.83     | 0.72              | 0.90              |

We now assess the extraction process using expert rules, find association rules for classification and evaluate it with the result produced by expert rules. We analyse the effect of

MA as part of extraction process. The definition of expert rules from Section 4.1 is applied to the transformed datasets (with and without MA) and the results are shown in Table 5. We can see that applying MA during data extraction, gives us the increments of 1% in Accuracy and Specificity while considering Maximum summary. For median and average summaries, Accuracies remain the same, however, Sensitivities improve while applying MA. For minimum summary, we get better results in all three metrics while applying MA. Therefore, the result justifies the use of moving average to the features.

Tab. 6: Classification results of mined rules.

| summary | Confidence | Accuracy | Sensitivity (TPR) | Specificity (TNR) |
|---------|------------|----------|-------------------|-------------------|
| Maximum | 0.709      | 0.70     | 0.96              | 0.43              |
|         | 0.712      | 0.72     | 0.92              | 0.52              |
|         | 0.715      | 0.77     | 0.85              | 0.69              |
|         | 0.724      | 0.79     | 0.83              | 0.76              |

For associative classification, we perform  $5 \times 5$  cross validation with the mined rules and the results are shown in Table 6. We consider the Maximum summary, as it gives the best result for expert rule in terms of Accuracy, Sensitivity and Specificity (see Table 5). We can see that Confidence 0.724 gives us better accuracy whereas Confidence 0.709 has lower accuracy but higher Sensitivity. In our work, we prioritize Sensitivity over Specificity. Sensitivity corresponds to the true positive rate for SIRS. Therefore, it is crucial that we do not miss a significant amount of diagnosis regarding SIRS. However, we also want the difference between Sensitivity and Specificity to be minimized. Therefore, Confidence 0.712 is more preferable, as Sensitivity and Specificity are more than 0.90 and 0.50, respectively.

Tab. 7: Top 10 generalized rules (N=14920, Rules=103, Min.confidence=0.71).

| LHS   | RHS  | Confidence |
|---|------|------------|
| Leukocyte high, Respiration high, Temperature low   |      | 1.0        |
| Leukocyte high, Pulse tachycardia, Respiration high |      | 1.0        |
| Leukocyte high, Respiration high, Temperature high  |      | 1.0        |
| Leukocyte low, Respiration high                     |      | 1.0        |
| Leukocyte high, Respiration high                    | SIRS | 0.971      |
| Leukocyte high, Pulse tachycardia                   |      | 0.967      |
| Respiration high, Temperature low                   |      | 0.923      |
| Leukocyte high, Temperature high                    |      | 0.913      |
| Pulse normal, Respiration high, Temperature high    |      | 0.909      |
| Respiration high, Temperature high                  |      | 0.905      |

Figure 3 shows the frequency of the features or items in the itemsets. From this figure, we can see the importance of the features which is crucial for the validation of the SIRS criteria. In the SIRS criteria, abnormal leukocyte count and temperature is given higher importance as mentioned in Section 4.1. From Figure 3, we can see, both features are at the top. The third, fourth, and fifth features are pulse tachycardia, high respiration and normal diastolic

pressure, respectively. Naturally, the normal categories are in the bottom. Normal diastolic pressure is in the fourth place, which tells us that in many SIRS cases diastolic pressure remains normal when other parameters are abnormal (either high or low). We discover rules for SIRS patients using ARM techniques and the top 10 mined rules are shown in Table 7. Our study reports a relatively higher proportion of abnormal Leukocyte count, temperature and pulse in SIRS patients. Applying these mined rules to the transformed dataset gives us better results than the SIRS criteria in terms of Sensitivity. As we present the top 10 rules, most of the rules are aligned with the SIRS criteria which also validates the criteria itself.

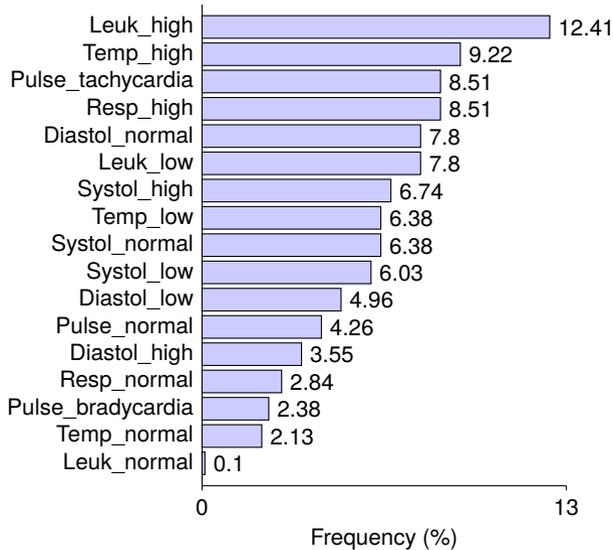


Fig. 3: Frequency of vital signs and lab test (items in the itemsets).

Better accuracy does not imply better results. From the experiment, we see that the expert rules have an accuracy at most 0.87 (Table 5). However, for that accuracy, we have a Sensitivity around 0.76 and Specificity around 0.93. As we prioritize Sensitivity over Specificity, the result of the mined rules with Maximum summary is more favorable. It can also be noted that the Sensitivity (TPR) and the Specificity (TNR) have a reciprocal relationship. With increasing confidence, Sensitivity (TPR) reduces whereas Specificity (TNR) escalates. Therefore, we can easily choose a confidence that can give us Sensitivity over 0.9 and Specificity over 0.5.

## 6 Conclusion

This work demonstrated a data mining technique in terms of feature extraction and transformation for finding association rules using Apriori algorithm. These rules are used to perform Associative Classification and compared with the results found using SIRS criteria.

Using SIRS criteria on the transformed dataset also showed the quality of the extraction process. We analyzed the effect of Moving Average (MA); it was found that MA improved the result to some extent in terms of Accuracy, Sensitivity and Specificity when applying SIRS criteria. In simple terms, we extracted a feature set with four summaries (Maximum, Minimum, Median and Average). Then we applied the SIRS criteria with a set of thresholds for age-specific vital signs and laboratory variables from the literature to see which summary is better. Then we used that summary for rule mining. After generating a set of rules, we again applied the mined rules for classification. This experiment was carried out to show the efficacy of mined rules while evaluated with expert rules (SIRS criteria).

Due to the nature of the diagnosis, Sensitivity was given more priority than Specificity and they showed a reciprocal relationship. By tuning the confidence of the mined rules, we gained control over Sensitivity and Specificity. From the SIRS criteria, the best result was found with Accuracy, Sensitivity and Specificity up to 0.87, 0.76 and 0.93, respectively for the Maximum summary. While applying mined rules with varying Confidence, we were able to achieve higher Sensitivity up to 0.96. However, Accuracy and Specificity were reduced to 0.70 and 0.43, respectively. Therefore a Confidence with 0.712 was more suitable where the Accuracy, Sensitivity and Specificity were around 0.72, 0.92 and 0.52, respectively.

In future work, we plan to explore other ways of feature extractions and perform a comparative analysis with SIRS criteria. We will also apply different Machine Learning algorithms like Decision Tree, Random Forest, Support Vector Machine.

## Acknowledgments

The ELISE project is partially funded by the Federal Ministry of Health; Grant No. 2520DAT66A. This work was also partially supported by the Fraunhofer Internal Programs under Grant No. Attract 042-601000. Ethics approval for use of routine data was given by the Ethics Committee of Hannover Medical School (approval number 9819\_BO\_S\_2021). We would like to thank our colleagues from the MHH Information Technology (MIT) from the Hannover Medical School for their support.

## Bibliography

- [Aq19] Aqra, Iyad; Abdul Ghani, Norjihani; Maple, Carsten; Machado, José; Sohrabi Safa, Nader: Incremental algorithm for association rule mining under dynamic threshold. *Applied Sciences*, 9(24):5398, 2019.
- [AT14] Abdelhamid, Neda; Thabtah, Fadi: Associative classification approaches: review and comparison. *Journal of Information & Knowledge Management*, 13(03):1450027, 2014.
- [BN18] Borah, Anindita; Nath, Bhabesh: Identifying risk factors for adverse diseases using dynamic rare association rule mining. *Expert systems with applications*, 113:233–263, 2018.

- [Bo92] Bone, Roger C; Balk, Robert A; Cerra, Frank B; Dellinger, R Phillip; Fein, Alan M; Knaus, William A; Schein, Roland MH; Sibbald, William J: Definitions for sepsis and organ failure and guidelines for the use of innovative therapies in sepsis. *Chest*, 101(6):1644–1655, 1992.
- [Br57] Brown, Robert G: Exponential smoothing for predicting demand. In: *Operations Research*. volume 5. *Inst Operations Research Management Sciences*, pp. 145–145, 1957.
- [De06] De Lusignan, Simon; Metsemakers, Job; Houwink, Pieter; Gunnarsdottir, Valgerdur; VanDerLei, Johan: Routinely collected general practice data: goldmines for research? A report of the European Federation for medical informatics primary care informatics Working Group (EFMI PCIWG) from MIE2006, Maastricht, the Netherlands. *Journal of Innovation in Health Informatics*, 14(3):203–209, 2006.
- [Go05] Goldstein, Brahm; Giroir, Brett; Randolph, Adrienne et al.: International pediatric sepsis consensus conference: definitions for sepsis and organ dysfunction in pediatrics. *Pediatric critical care medicine*, 6(1):2–8, 2005.
- [Ha03] Han, Yong Y; Carcillo, Joseph A; Dragotta, Michelle A; Bills, Debra M; Watson, R Scott; Westerman, Mark E; Orr, Richard A: Early reversal of pediatric-neonatal septic shock by community physicians is associated with improved outcome. *Pediatrics*, 112(4):793–799, 2003.
- [Hy11] Hyndman, Rob J.: Moving Averages. In (Lovric, Miodrag, ed.): *International Encyclopedia of Statistical Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 866–869, 2011.
- [In09] Inwald, David P; Tasker, Robert C; Peters, Mark J; Nadel, Simon: Emergency management of children with severe sepsis in the United Kingdom: the results of the Paediatric Intensive Care Society sepsis audit. *Archives of disease in childhood*, 94(5):348–353, 2009.
- [Iv15] Ivančević, Vladimir; Tušek, Ivan; Tušek, Jasmina; Knežević, Marko; Elheshk, Salaheddin; Luković, Ivan: Using association rule mining to identify risk factors for early childhood caries. *Computer Methods and programs in Biomedicine*, 122(2):175–181, 2015.
- [Ka16] Kamalesh, Murari Devakannan; Prasanna, K Hema; Bharathi, B; Dhanalakshmi, R; Aroul Canessane, R: Predicting the risk of diabetes mellitus to subpopulations using association rule mining. In: *proceedings of the international conference on soft computing systems*. Springer, pp. 59–65, 2016.
- [Ke13] Keen, Justin; Calinescu, Radu; Paige, Richard; Rooksby, John: Big data+ politics= open data: The case of health care data in England. *Policy & Internet*, 5(2):228–243, 2013.
- [LNFV21] Liu, Xiangyu; Niu, Xinzhen; Fournier-Viger, Philippe: Fast top-k association rule mining using rule generation property pruning. *Applied Intelligence*, 51(4):2077–2093, 2021.
- [Na13] Nahar, Jesmin; Imam, Tasadduq; Tickle, Kevin S; Chen, Yi-Ping Phoebe: Association rule mining to detect factors which contribute to heart disease in males and females. *Expert Systems with Applications*, 40(4):1086–1093, 2013.
- [No18] Noguchi, Yoshihiro; Ueno, Anri; Otsubo, Manami; Katsuno, Hayato; Sugita, Ikuto; Kanematsu, Yuta; Yoshida, Aki; Esaki, Hiroki; Tachi, Tomoya; Teramachi, Hitomi: A new search method using association rule mining for drug-drug interaction based on spontaneous report system. *Frontiers in pharmacology*, 9:197, 2018.

- [RN20] Ramasamy, S; Nirmala, K: Disease prediction in data mining using association rule mining and keyword based clustering algorithms. *International Journal of Computers and Applications*, 42(1):1–8, 2020.
- [Rü22] Rübsamend, Nicole; Böhnked, Julia; Karchd, André; Dase, Pronaya Prosun; Wiese, Lena; Groszewski-Andersf, Christian; Hallerf, Andreas; Frankf, Torsten: Towards an evolutionary open pediatric intensive care dataset in the ELISE project. *Advances in Informatics, Management and Technology in Healthcare*, 295:100, 2022.
- [So17] Sonet, KM Mehedi Hasan; Rahman, Md Mustafizur; Mazumder, Pritom; Reza, Abid; Rahman, Rashedur M: Analyzing patterns of numerously occurring heart diseases using association rule mining. In: 2017 twelfth international conference on digital information management (ICDIM). IEEE, pp. 38–45, 2017.
- [Ta07] Tarca, Adi L; Carey, Vincent J; Chen, Xue-wen; Romero, Roberto; Drăghici, Sorin: Machine learning and its applications to biology. *PLoS computational biology*, 3(6):e116, 2007.
- [Wu19] Wulff, Antje; Montag, Sara; Steiner, Bianca; Marschollek, Michael; Beerbaum, Philipp; Karch, André; Jack, Thomas: CADDIE2—evaluation of a clinical decision-support system for early detection of systemic inflammatory response syndrome in paediatric intensive care: study protocol for a diagnostic study. *BMJ open*, 9(6):e028953, 2019.
- [Wu21] Wulff, Antje; Montag, Sara; Rübsamen, Nicole; Dziuba, Friederike; Marschollek, Michael; Beerbaum, Philipp; Karch, André; Jack, Thomas: Clinical evaluation of an interoperable clinical decision-support system for the detection of systemic inflammatory response syndrome in critically ill children. *BMC medical informatics and decision making*, 21(1):1–9, 2021.
- [XJ19] Xu, Chunming; Jackson, Scott A: Machine learning and complex biological data. *Genome biology*, 20(1):1–4, 2019.

# SportsTables: A new Corpus for Semantic Type Detection

Sven Langenecker,<sup>1</sup> Christoph Sturm,<sup>2</sup> Christian Schalles,<sup>3</sup> Carsten Binnig<sup>4</sup>

**Abstract:** Table corpora such as VizNet or TURL which contain annotated semantic types per column are important to build machine learning models for the task of automatic semantic type detection. However, there is a huge discrepancy between corpora that are used for training and testing since real-world data lakes contain a huge fraction of numerical data which are not present in existing corpora. Hence, in this paper, we introduce a new corpus that contains a much higher proportion of numerical columns than existing corpora. To reflect the distribution in real-world data lakes, our corpus SportsTables has on average approx. 86% numerical columns, posing new challenges to existing semantic type detection models which have mainly targeted non-numerical columns so far. To demonstrate this effect, we show the results of a first study using a state-of-the-art approach for semantic type detection on our new corpus and demonstrate significant performance differences in predicting semantic types for textual and numerical data.

**Keywords:** Semantic Type Detection; Column Annotated Corpora

## 1 Introduction

**Semantic type detection is important for data lakes.** Semantic type detection of table columns is an important task to exploit the large and constantly changing data collections residing in data lakes. However, manually annotating tables in data lakes comes at a high cost. Hence, in the past many approaches have been developed that automatically derive semantic types from table data [Hu19b, Zh20, De21, Su22]. Many of the recent approaches use deep learning techniques to build semantic type detection models. As such, corpora containing large amounts of table data with assigned semantic types are required for training and validating. Existing annotated table corpora (e. g. VizNet, TURL) primarily contain tables extracted from the web and therefore limit the capability to represent enterprise data lakes.

**Existing corpora and models fall short on real-world data lakes.** However, as we can see in Fig. 1, almost all existing corpora that provide annotated columns labeled with semantic types have a lack of table columns that contain numerical data, and tables in these datasets incorporate either only or a very high percentage of textual data. Only GitTables [HDG21] contains a more balanced ratio of textual and numerical data. Nevertheless, compared to real

---

<sup>1</sup> DHBW Mosbach, Germany sven.langenecker@mosbach.dhbw.de

<sup>2</sup> DHBW Mosbach, Germany christoph.sturm@mosbach.dhbw.de

<sup>3</sup> DHBW Mosbach, Germany christian.schalles@mosbach.dhbw.de

<sup>4</sup> TU Darmstadt, Germany carsten.binnig@tu-darmstadt.de

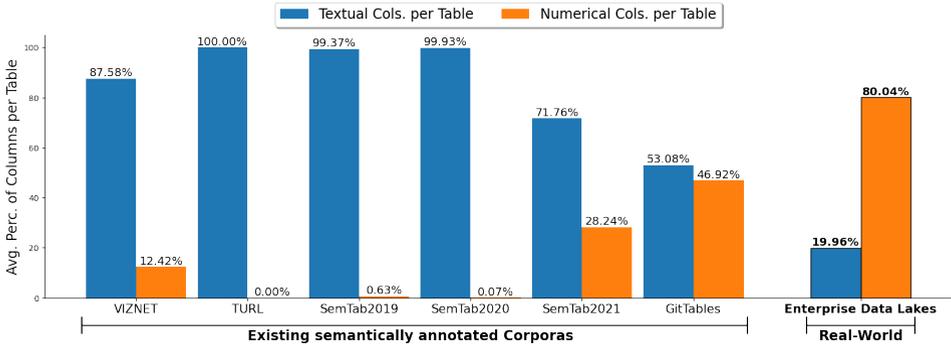


Fig. 1: Average percentage of textual and numerical based columns per table in existing semantically annotated corpora<sup>6</sup> (left bars) compared to real-world data lakes (right bar). This shows the fact that there is a significant shift in the ratio of textual to numeric columns per table from existing corpora to real data lakes. Since all existing semantic type detection models were developed by using the existing corpora, shortcomings in validating the models on numerical data are present and it has not yet been studied in depth how well the models can perform on datasets containing a high proportion of numerical data.

enterprise data lakes, there is a significant discrepancy in the ratio of textual to numerical data. An inspection of a large real-world data lake at a company<sup>5</sup> has shown that on average approx. 20% textual data and 80% numerical data are present (see. Fig. 1 bars on the right). Moreover, semantic type detection models [Hu19b, Zh20, De21, Su22] that are trained on the available corpora also mainly target non-numerical data.

**Semantic type detection for numerical data is challenging.** Detecting semantic types of numerical columns is generally harder than for textual columns. For example, for a textual column with the values {Germany, USA, Sweden, ...} a model can easily identify the semantic type *country*. Instead, for a numeric column with e. g. the values {20,22,30,34,...} it is not that straightforward and several possibilities for a matching semantic type exist such as *age*, *temperature*, *size*, *money*. The fundamental reason here is that numerical values can be encoded with much fewer bits than string values [Sh48], resulting in a lower overall entropy and thus providing less information content that can be used by a machine learning model to infer the underlying semantic type. Due to the existing corpora providing annotated columns that have been used to create and validate semantic type detection models, we see several essential shortcomings that could not be addressed until now because of the absence of a sufficient dataset for this purpose.

<sup>5</sup> The analyses were done at the company LÄPPLER AG

<sup>6</sup> Notice that for GitTables we only considered the tables and columns labeled by terms from DBpedia using the semantic annotation method as described in the GitTables paper. Therefore our reported ratios of textual and numerical data differ from those shown in the GitTables paper because they consider all data, whether annotated or not.

**Contributions.** In this paper, we thus contribute a new corpus containing tables with semantically annotated columns with numeric and non-numeric columns that reflect the distribution of real-world data lakes. We will make the corpus available which should stimulate research directions such as working on new model architectures that can reliably annotate types to numeric and non-numerical columns. In the following, we discuss the main contribution of this paper.

As a first contribution, we present and provide our new corpus SportsTables<sup>7</sup>. To the best of our knowledge, SportsTables is the first corpus with annotated table columns, which contains a significantly larger proportion of numerical data than textual data. In total, the tables in our corpus have on average about 3 textual and 18 numerical columns. Moreover, the tables in our new corpus are much larger in both the number of columns and the number of rows than in existing corpora which better reflects the characteristics of real-world tables.

As a second contribution that comes together with the corpus, we specify an ontology with semantic types for the sports baseball, basketball, football, hockey, and soccer. This ontology provides fine granular semantic types for all kinds of sports we considered to build SportsTables and allows us to semantically describe each occurring table column, which is not possible with the current ontologies (e. g. DBpedia) at this level of detail. Using a manually created dictionary, we assign a semantic type to each existing column in SportsTables.

As a third contribution, we present our initial results of using our new corpus on Sato [Zh20], a state-of-the-art semantic type detection model. Overall, we can see that when trained on our new corpora, Sato can improve the performance on numerical data types. However, one shortcoming that our analysis shows is that current model architectures are not targeting numerical columns. To be more precise, our analysis demonstrates that textual data columns are mostly correctly semantically interpreted with Sato (F1-Score of 1.0), but on numerical data columns, the model only achieves an F1-Score of about 0.55. This large difference indicates that new model architectures that take the characteristics of numerical columns into account are needed which is a direction that could be stimulated by the availability of our corpus.

**Outline.** In Section 2, we first provide an overview of existing corpora which was used to build and validate semantic type prediction models and discuss their characteristics and statistics. Afterward, in Section 3, we then introduce our new corpus SportsTables and describe in detail how we created the corpus and labeled the table columns with semantic types. Section 4 first demonstrates the main characteristics of our corpus before we then show the initial results of using our new corpus on Sato. Next, further research challenges are discussed in Section 5 before Section 6 concludes the paper.

---

<sup>7</sup> Available on <https://github.com/DHBWMosbachWI/SportsTables.git>

| Corpus              | #Table       | #Total Columns | Avg. #Columns per Table | Avg. #Rows per Table |
|---------------------|--------------|----------------|-------------------------|----------------------|
| VIZNET              | 78,733       | 120,609        | 1.53                    | 18.35                |
| TURL                | 406,706      | 654,670        | 1.61                    | 12.79                |
| SemTab2019          | 13,765       | 21,682         | 1.58                    | 35.61                |
| SemTab2020          | 131,253      | 190,494        | 1.45                    | 9.19                 |
| SemTab2021          | 795          | 3,072          | 3.86                    | 874.6                |
| GitTables           | 1.37M        | 9.3M           | 6.82                    | 184.66               |
| <b>SportsTables</b> | <b>1,187</b> | <b>24,838</b>  | <b>20.93</b>            | <b>246.72</b>        |

Tab. 1: Corpus statistics about the number and sizes of tables.

## 2 Existing Corpora with Semantic Data Types

In the following, we describe different existing corpora that contain annotated table columns and therefore can be used to build and validate semantic column type detection models. We summarized the main statistics for all corpora in Tab. 1.

**VizNet [Hu19a].** The original VizNet corpus [Hu19a] is a collection of data tables from diverse web sources ([Ca08, Vi07, P118, NUP16]) which initially do not contain any semantic label annotation. The corpus we consider in this paper is a subset of the original VizNet corpus, which was annotated by a set of mapping rules from column headers to semantic types and then used to build and validate the Sherlock [Hu19b] and Sato[Zh20] prediction model. The corpus contains in total 78,733 tables and 120,609 columns annotated with 78 unique semantic types. Overall, the tables in the corpus contain only 1.53 columns and 18.35 rows on average. Furthermore, the distribution of the column data types is 87.58% textual and 12.42% numerical and thus leads to the shortcomings as described before.

**TURL [De21].** The TURL corpus uses the WikiTable corpus [BND15] as basis. To label each column they refer to the semantic types defined in the Freebase ontology [Go22] with a total number of 255 different semantic types. What distinguishes TURL from other corpora is that columns can have multiple semantic types assigned. In total, there are 406,706 tables resulting in 654,670 columns, and on average a table consists of 1.61 columns and 12.79 rows. Again, these are rather small dimensions. In addition, the Turl corpus includes no numerical data at all, which leads to the shortcomings mentioned above when using the corpora.

**SemTab.** SemTab is a yearly challenge with the goal of benchmarking systems that match tabular data to knowledge graphs since 2019. The Challenge includes the tasks of assigning a semantic type to a column, matching a cell to an entity, and assigning a property to the relationship between columns. Every year, the challenge provides different datasets to validate the participating systems against each other. In this paper we observed the provided corpora for the years 2019 [Ha19], 2020 [Ha20, Cu20], and 2021 [Ha21, Cu20, OP21, ASKR21, HDD21]. Statistic details of the corpora are shown in Tab. 1. In case more than one dataset was provided per year, we aggregated the statistics over all datasets included in the challenge. While SemTab2019 consists of 13,765 tables and 21,682 columns in

total, there are 131,253 tables and 190,494 columns in SemTab2020. In both corpora, the dimensions of the included tables are rather small (on average 1.58 columns and 35.61 rows in 2019 and 1.45 columns and 9.19 rows in 2020). In SemTab2021, the contained tables are the largest in terms of rows with almost 875 on average. However, the number of columns (3.86 on average) is only moderate and the corpus in general is the smallest with a total of 795 tables and 3,072 columns. Numerical data is almost nonexistent in the first two years (0.63% in 2019 / 0.07% in 2020), increasing to 28.24% numeric columns per table on average in 2021, which is still not comparable to the number of numeric data in real world data lakes.

**GitTables [HDG21].** GitTables is a large-scale corpus of relational tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [GBM16] and DBpedia [Au07] using two different automated annotation methods (syntactically/semantically similarity matching from semantic type to column header). In this paper, we have focused on the annotations origin from DBpedia and the results of the semantic annotations method as described in the GitTables paper [HDG21]. This leads to a corpus containing over 1.37M tables and 9.3M columns in total. Although this is by far the largest collection of data tables, the dimensions of the tables are on average only moderate with 6.82 columns and 184,66 rows. Overall, GitTables incorporates the most numeric data with an almost balanced ratio of 53.08% textual and 46.92% numerical columns per table.

**Discussion.** The overview in Tab. 1 and the discussion before shows that most existing corpora contain no or only a minimal fraction of numerical data types which is very different from real-world data lakes. An exception is GitTables which has a much higher ratio of numerical columns. However, as we show in Sect. 4, GitTables still lacks a good coverage of different numeric semantic types which is one important aspect that we tackle with our new corpus SportsTables which covers a wide variety of different numerical semantic types. Moreover, another important (but orthogonal) aspect is that existing corpora include a large number of tables. However, on average the tables are very small in terms of the number of columns and the number of rows. Instead, our new corpus SportsTables contains fewer tables, but on average a significantly higher number of columns and rows per table to better reflect the characteristics of real-world data lakes

### 3 The SportsTables Corpus

In the following, we will introduce our new corpus and describe in detail the implemented construction pipeline to build SportsTables.

**Methodology to generate corpus.** Fig. 2 gives an overview of our implemented pipeline to generate the new corpus. The main idea was to collect data tables from different sports domains such as soccer, basketball, baseball, etc. since data tables coming from such kinds of sources are rich in numerical columns. For example, a soccer player statistic table of

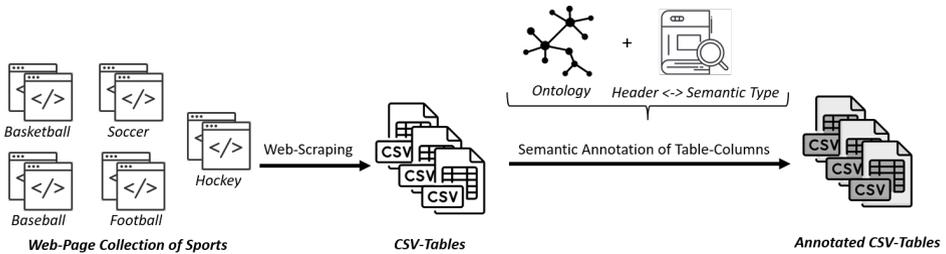


Fig. 2: Overview of the implemented pipeline to build SportsTables. We use web-scraping techniques to extract HTML tables from a manually defined web page collection for each selected sport and convert the tables to CSV files. With the help of a defined ontology and a manually created dictionary that maps column headers to semantic types, we annotate each table column with an appropriate semantic type.

a soccer season contains typically 3 textual columns (e. g., player name, team name, field position) and 18 numerical columns (e. g., goals, games played, assists). Hence, building a collection of such tables will lead to a corpus that contains many numerical columns which are in addition semantically interpretable. As a result, the corpus will enable to analyze the performance of semantic type prediction models in a much more rigorous manner regarding numerical data.

**Scraping data from the web [Di19].** A vast amount of data covering information about player statistics, team statistics, coach statistics, or season rankings of different sports are available on various web pages. Therefore, for collecting the data, we built a data collection pipeline based on web scraping technology[Di19]. In the first step, we manually searched and defined a set of different web pages for each of the selected sports of which we want to scrape contained data tables (left side of Fig. 2). We first converted each HTML table on the web pages to Pandas-Dataframes using Python and then saved them as CSV files (center of Fig. 2), since this file format is most known and used to store raw structured data [Mi16]. During the scrape process, we kept the respective column headers from the original HTML table and used them as headers in the CSV file.

**Annotating columns with semantic types.** Due to the low granularity of existing ontologies (e. g. DBpedia) regarding semantics of a given sport, we manually created an ontology-like set of valid semantic types for all sports. For example, in DBpedia there is the type *Person.Athlete.BasketballPlayer*, but semantic labels in the particular that would match individual numerical columns such as *NumberOfGoals* are not defined. Next, we annotated all table columns with semantic types using a manually created dictionary that maps column headers to matching semantic types from our created set. Since the column headings were in many cases identical if the semantic content was the same, this procedure significantly reduces the manual labeling effort. In addition, to ensure that the labels are of very high quality in terms of correctness, we manually checked each assignment based on the content of the columns.

| Sports     | #Table | #Total Cols | Avg. #Text. Cols per Table | Avg. #Num. Cols per Table | Avg. #Rows per Table |
|------------|--------|-------------|----------------------------|---------------------------|----------------------|
| Baseball   | 174    | 3,829       | 3.97                       | 18.03                     | 76.34                |
| Basketball | 180    | 3,801       | 1.78                       | 19.34                     | 152.5                |
| Football   | 303    | 6,764       | 2.45                       | 19.88                     | 354.79               |
| Hockey     | 257    | 5,347       | 2.1                        | 18.7                      | 247.15               |
| Soccer     | 273    | 5,097       | 3.9                        | 14.77                     | 297.11               |
| Total      | 1,187  | 24,838      | 2.83                       | 18.1                      | 246.72               |

Tab. 2: Corpus statistics about the number and size of included tables. Statistics are shown broken down by individual sports taken into account and in total. Across all sports, the average number of numeric columns is much higher than textual columns.

## 4 Analysis of the Corpus

This section describes the characteristics of SportsTables in detail and then demonstrates the significant impact of these characteristics on semantic type prediction frameworks in a small study where we apply the corpus to an existing type detection model.

### 4.1 Corpus Characteristics

In the following, we discuss the statistics of the SportsTables corpus and compare them to the existing corpora.

**Data statistics (Tab. 1&Tab. 2).** Using the described pipeline for creating SportsTables, a total of 1,187 tables which comprises 24,838 columns (approx. 86% numeric and 14% textual) are scraped from the web resulting in 20.93 columns (2.83 textual and 18.1 numerical) per table on average. This ratio of textual to numerical columns, as well as the total average number of columns in a table, differs significantly from existing corpora. To provide details about the contribution of different sports areas contained in SportsTables, Tab. 2 shows the main statistics by the individual areas of sports.

Fig. 3 shows a comparison of the average number of textual and numerical columns per table of SportsTables versus that of the existing corpora. Here we can see that numerical columns only exist in the corpora VizNet with 0.33, SemTab2021 with 1.09, and GitTables with 3.2 columns per Table. Compared to GitTables, in SportsTables there are thus on average over 6 times more numeric columns per table. Moreover, as we discuss below, our corpus uses a much richer set of numerical data types that better reflects the characteristics in real-world data lakes which is very different from GitTables. For example, when looking at the semantic types that are assigned to numerical columns in GitTables, more than half (393,925) of the columns are labeled with just a single type *Id*.

In terms of the total number of columns, the tables in SportsTables (20.93 columns per table) are on average about 3 times wider than in GitTables (6.82 columns per table), which contains the widest tables among the existing corpora. As such, the number of columns in tables

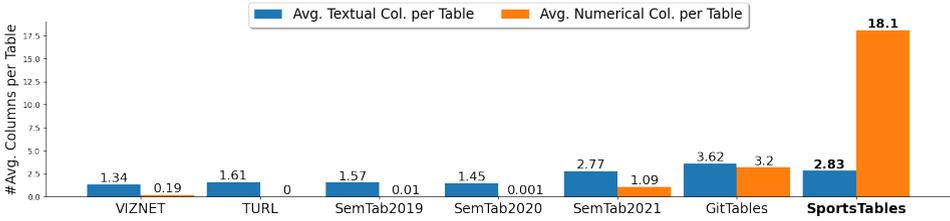


Fig. 3: Average number of textual and numerical columns per table for each existing annotated corpora and our new SportsTables corpus. This shows the absence of numerical data columns per table in most existing corpora and the dominance of textual data columns per table in all existing corpora. Instead, our new corpus SportsTables contains on average over 6 times more numerical columns than textual columns.

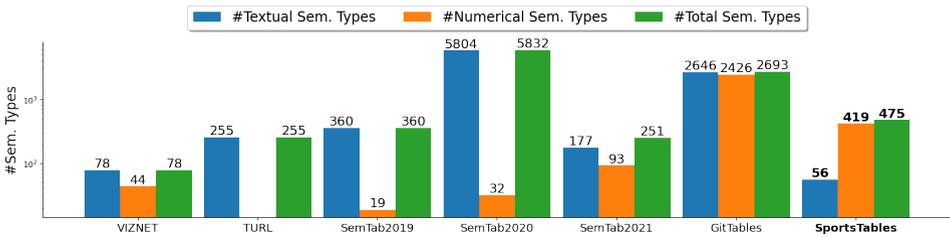


Fig. 4: Corpus statistics about the number of unique semantic types included. Showing that our new corpus has a higher proportion of numerical semantic types than textual semantic types in contrast to the existing corpora. In addition, there is a large overlap of semantic types used for textual and numeric columns in the existing corpora. In comparison, the semantic types in SportsTables are disjoint for the two column data types.

of SportsTables are reflecting better the width when comparing this to the characteristics of the tables in the real-world data lakes which we analyzed. Moreover, considering the average number of rows per table, it can be seen that the tables in SportsTables have on average 246.72 rows. In comparison, tables in SportsTables are larger on average than in many other corpora where tables have typically fewer rows.

**Annotation statistics.** Semantic type annotation follows a two step process. First, we establish a directory with manually defined mappings from column header to semantic type for each existing header. Second, we label each column with the semantic type listed in the directory for its header. As a result, 56 textual and 419 numerical semantic types are present in the corpus. Thereby textual semantic types are those which specify textual columns and numerical types are those which specify columns containing numeric values. To compare the annotation statistics, we also counted the number of textual and numerical semantic types in an analysis of the existing corpora. The results of these analyses can be seen in Fig. 4. Different from our corpus, the sets of textual and numerical types are not disjoint in all other corpora (except TURL where no numeric values are present). This indicates that

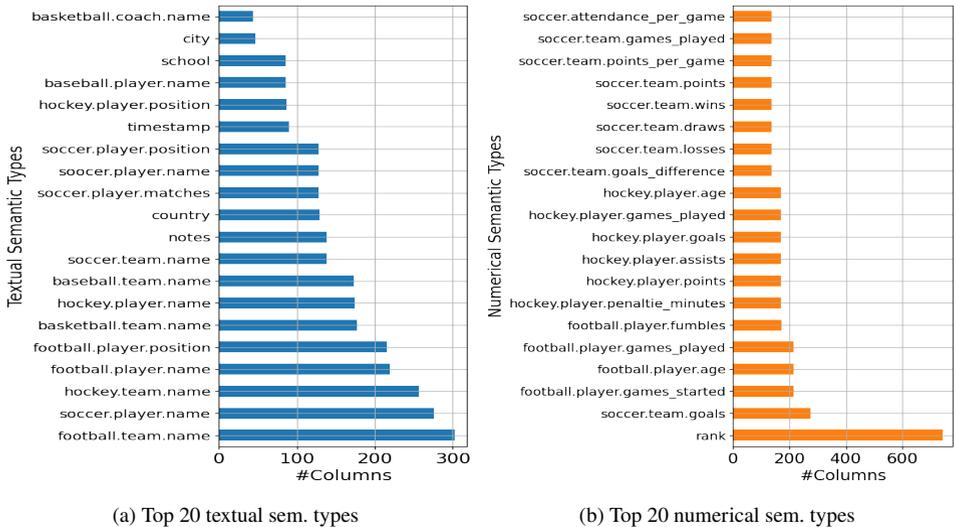


Fig. 5: Semantic type annotation statistics of SportsTables. (a) Shows column annotation counts of the top 20 textual semantic types. Across all kinds of sports, *player.name* and *team.name* are the most common. (b) Shows column annotation counts of the top 20 numerical semantic types. A dominant type here is *rank*, which describes a column containing the placements of e. g. a team in a season standings table.

individual semantic types were assigned to both textual and numerical columns which is problematic if semantic type detection models should be trained and tested on these corpora. In particular, GitTables has a very large overlap and almost all semantic types are used in both column data types. To give an example, in GitTables the semantic types *comment*, *name* and *description* are assigned to both column data types. Next, we take a closer look into the semantic types of our corpus.

Fig. 5a and Fig. 5b shows the top 20 semantic types (textual and numerical) in regards to how often they were assigned to a table column. It can be seen that the most common textual types across all sports are *player.name* and *team.name*. These are types that occur in almost every table. Other types such as *country* or *city* are also common, describing, for example, the player's origin or the team's hometown. Among numeric semantic types, *rank* is by far the most common and is present in almost all tables. The type describes a column containing the placement of e. g., a team in a "seasons standing" table or a player in a "top scorer" table. All other numeric semantic types show mainly an equal distribution of the frequency, which is a good precondition for training machine learning models. In order to show not only the frequency of the top 20 semantic types, Fig. 6 plots all semantic types (separated in textual and numerical) by the frequency of occurrences. Here we see that 19 textual and 66 numerical semantic types occur only once in the entire corpus. For

the training and testing of prediction models, we would suggest not considering these types due to the low occurrences.

**SportsTables vs. GitTables.** Since GitTables is the largest corpus with the most tables, one could argue that a subset of GitTables would result in a new corpus with similar characteristics as SportsTables. To analyze this, we executed a small experiment in which we filtered out only tables from GitTables where the number of textual and numerical columns (min. 3 textual and 18 numerical columns) is at least the same as it is in SportsTables. The result was a corpus containing a total of 16,909 tables and 743,432 columns. On average a table has 12.53 textual columns, 31.43 numerical columns, and 17.35 rows. However, looking at the semantic types that are assigned to numerical columns, more than half (393,925) of the columns are labeled with the type *Id*. In terms of training and validating semantic type detection models, this is rather an unfavorable type representing no semantically meaning. Moreover, the next 5 most common numerically based semantic types are *parent*, *max*, *comment*, *created* and *story editor*, constituting a large proportion of the columns. The assignment of these types to numerical data is slightly less understandable and indicates a lack of quality in the automatically generated labels for table columns.

## 4.2 An Initial Study of Using SportsTables

In the following, we report on the initial results of using Sato, a recent semantic type detection model, on our new corpus. With this, we want to measure how well the semantic types in our corpus can be inferred by the model with a special focus on how it performs on textual and numerical columns.

**Experiment setup.** For the first experiments, we only considered the soccer data from SportsTables. Thereby, we split the corpus into different sizes of train and test sets (5/95, 10/90, 15/85, 20/80), to show the results of scenarios where the model has less and more training data available. We use the pre-trained Sato model, which was trained on the VizNet corpus, and re-trained it with the different training set sizes. During re-training, we replaced the last layer of Sato to support the number of semantic types that occur in SportsTables and then re-trained the entire neural network. To measure the performance, we applied the re-trained model to the corresponding test data set.

**Results of study.** Fig. 7 shows the results of the experiments reporting F1-Scores using the defined different sizes of train and test splits as described before. We plot macro and weighted average F1-Score across all semantic types to show the total performance, but also separate average F1-Score for only textually and numerically based semantic types, respectively. As we can see in the figure, while Sato can detect numerical types, there is a significant performance difference between predicting textual and numerical semantic types for all setups. At the data split 20/80, all textual columns can be predicted correctly by the model, whereas for numerical columns only an F1-Score of 0.56 is achieved. On average, the difference in F1-Score between textual and numeric types is 0.41 across all setups.

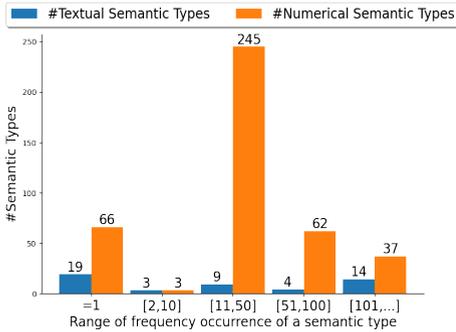


Fig. 6: Shows how often semantic types occur in SportsTables using buckets of varying widths, which represents the frequency of occurrences. For example, 19 textual and 66 numerical types occur only once in the entire corpus.

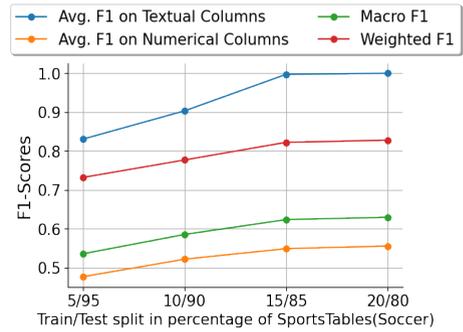


Fig. 7: Initial results using the Sato model on our new SportsTables corpus with different train/test split sizes. The differences in F1-Scores for predicting textual and numeric columns indicate that the model can handle textual data more effectively than numeric data.

These results indicate that the model is more able to handle textual data and determine the associated semantic type more accurately than for numeric data. Across all semantic types, the weighted F1-Score increase from 0.73 to 0.82 while the macro F1-Score range from 0.53 to 0.63, which are rather moderate score values for semantic type prediction models.

## 5 Further Research Challenges

Detecting semantic types in real-world data lakes comes with many more challenges that need to be addressed. In particular, based on our findings of the analysis using Sato in Section 4, we think that new model architectures are needed for detecting numerical data types which have very different characteristics from non-numerical data. In the following, we list some of the challenges we think are important to be addressed. We hope that our corpus enables research on those challenges.

*Embedding numerical data:* Most state-of-the-art models apply language models like BERT [De19] to encode literals to infer the semantic type of a table column. Since such approaches are optimized for textual data, the performance on numerical data of such models is not entirely analyzable with the existing corpora.

*Leveraging numerical context:* To improve the semantic type prediction of a table column, recent approaches like Sato [Zh20], TURL [De21] and Doduo [Su22] incorporate also context information like the table-topic or values from neighboring columns of the same table. Given that tables in existing corpora contain almost entirely textual columns, the contexts (e. g. values from neighboring columns) used are rich in information and therefore also lead to performance improvements. However, it is unclear how effective this approach

is in case the tables contain many numerical columns and only a few textual columns since the context information provided is reduced due to the lower entropy of numeric values as described before.

*Supporting wide tables:* Existing datasets for semantic type detection consist of tables with small numbers of columns and rows. In nearly all corpora, the existing tables contain on average less than two columns and less than 40 rows (see Tab. 1). Therefore, at the current state, it has not been analyzed how state-of-the-art models can handle such large tables. To give an example of why large tables could be a problem for recent models, we will briefly discuss Doduo[Su22]. Doduo uses pre-trained language models (e. g., BERT) and hence they have to convert the entire table into token sequences with a fixed tensor length of 512 elements so that the table and its entries can be meaningfully processed by the language model. To accomplish this, Doduo serializes the complete table and its entries as follows: for each table that has  $n$  columns  $T = (c_i)_{i=1}^n$ , where each column has  $N_m$  column values  $c_i = (v_i^j)_{j=1}^{N_m}$ , they let  $serialize(T) ::= [CLS]v_1^1 \dots [CLS]v_1^{N_m} \dots v_m^1 \dots v_m^{N_m} [SEP]$ , where the special token [CLS] marks the beginning of a new table column and [SEP] the end of a token sequence. With this methodology of serialization and the fixed given tensor length, increasing the number of table columns means that decreasing number of values of each column can be included for serialization. For example, a table with 512 columns would allow only one value per column to be considered and this would most likely result in an insufficient semantic representation of the column based on that one value.

## 6 Conclusion

Existing corpora for training and validating semantic type detection models mainly contain tables with only or a very high proportion of textual data columns and no or just a limited number of numerical data columns. Therefore, it has not been studied precisely how well state-of-the-art models perform on a dataset with a very high percentage of numerical columns as it occurs in real-world data lakes. Moreover, tables in existing corpora are very small regarding the total number of columns and rows. To tackle these shortcomings, we built a new corpus called SportsTables which contains tables that have on average approx. 3 textual columns, 18 numerical columns, and 250 rows. With our new corpus, semantic type detection models for table columns can now be holistically validated against numerical data. We show initial results by using Sato – a state-of-the-art model – on our new corpus and report significant differences in the performance of predicting semantic types of textual data and numerical data. The corpus is available on <https://github.com/DHBWMosbachWI/SportsTables.git>. Finally, we think that the corpus is just a first step to stimulate more research on new model architectures that can better deal with numerical and non-numerical data types.

## Bibliography

- [ASKR21] Abdelmageed, Nora; Schindler, Sirko; König-Ries, Birgitta: , fusion-jena/BiodivTab, October 2021. <https://doi.org/10.5281/zenodo.5584180>.
- [Au07] Auer, Sören; Bizer, Christian; Kobilarov, Georgi; Lehmann, Jens; Cyganiak, Richard; Ives, Zachary: DBpedia: A Nucleus for a Web of Open Data. In (Aberer, Karl; Choi, Key-Sun; Noy, Natasha; Allemang, Dean; Lee, Kyung-II; Nixon, Lyndon; Golbeck, Jennifer; Mika, Peter; Maynard, Diana; Mizoguchi, Riichiro; Schreiber, Guus; Cudré-Mauroux, Philippe, eds): *The Semantic Web*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 722–735, 2007.
- [BND15] Bhagavatula, Chandra Sekhar; Noraset, Thanapon; Downey, Doug: TabEL: Entity Linking in Web Tables. In: *The Semantic Web - ISWC 2015*. Springer International Publishing, Cham, pp. 425–441, 2015.
- [Ca08] Cafarella, Michael J.; Halevy, Alon; Wang, Daisy Zhe; Wu, Eugene; Zhang, Yang: WebTables: Exploring the Power of Tables on the Web. In: *VLDB*. volume 1. VLDB Endowment, p. 538–549, 2008.
- [Cu20] Cutrona, Vincenzo; Bianchi, Federico; Jiménez-Ruiz, Ernesto; Palmonari, Matteo: , Tough Tables: Carefully Evaluating Entity Linking for Tabular Data, November 2020. GT and Target files have different formats in 2T and 2T\_WD (CEA). 2T complies with the SemTab2019 CEA format (tab\_id, col\_id, row\_id, entity), while 2T\_WD follows the SemTab2020 CEA format (tab\_id, row\_id, col\_id, entity). Visit the SemTab challenge website (<https://www.cs.ox.ac.uk/isg/challenges/sem-tab/>) for more details.
- [De19] Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *NAACL-HLT 2019*. Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186, June 2019.
- [De21] Deng, Xiang; Sun, Huan; Lees, Alyssa; Wu, You; Yu, Cong: TURL: Table Understanding through Representation Learning. In: *VLDB*. volume 14. VLDB Endowment, pp. 307 – 319, 2021.
- [Di19] Diouf, Rabiyaou; Sarr, Edouard Ngor; Sall, Ousmane; Birregah, Babiga; Bouso, Mamadou; Mbaye, Sény Ndiaye: Web Scraping: State-of-the-Art and Areas of Application. In: *International Conference on Big Data*. pp. 6040–6042, 2019.
- [GBM16] Guha, R. V.; Brickley, Dan; Macbeth, Steve: Schema.Org: Evolution of Structured Data on the Web. *Commun. ACM*, 59(2):44–51, jan 2016.
- [Go22] Google: , Freebase Data Dumps. <https://developers.google.com/freebase>, 2022.
- [Ha19] Hassanzadeh, Oktie; Efthymiou, Vasilis; Chen, Jiaoyan; Jiménez-Ruiz, Ernesto; Srinivas, Kavitha: , SemTab 2019: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets, October 2019. <https://doi.org/10.5281/zenodo.3518539>.
- [Ha20] Hassanzadeh, Oktie; Efthymiou, Vasilis; Chen, Jiaoyan; Jiménez-Ruiz, Ernesto; Srinivas, Kavitha: , SemTab 2020: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets, November 2020. <https://doi.org/10.5281/zenodo.4282879>.

- [Ha21] Hassanzadeh, Oktie; Efthymiou, Vasilis; Chen, Jiaoyan; Jiménez-Ruiz, Ernesto; Srinivas, Kavitha; , SemTab 2021: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets, November 2021. <https://doi.org/10.5281/zenodo.6154708>.
- [HDD21] Hulsebos, Madelon; Demiralp, Çağatay; Demiralp, Paul; , GitTables benchmark - column type detection, November 2021. <https://doi.org/10.5281/zenodo.5706316>.
- [HDG21] Hulsebos, Madelon; Demiralp, Çağatay; Groth, Paul; GitTables: A Large-Scale Corpus of Relational Tables. CoRR, abs/2106.07258, 2021.
- [Hu19a] Hu, Kevin; Gaikwad, Snehal Kumar 'Neil' S.; Hulsebos, Madelon; Bakker, Michiel A.; Zraggen, Emanuel; Hidalgo, César; Kraska, Tim; Li, Guoliang; Satyanarayan, Arvind; Demiralp, Çağatay; VizNet: Towards A Large-Scale Visualization Learning and Benchmarking Repository. In: CHI '19. CHI '19, ACM, New York, NY, USA, p. 1–12, 2019.
- [Hu19b] Hulsebos, Madelon; Hu, Kevin; Bakker, Michiel; Zraggen, Emanuel; Satyanarayan, Arvind; Kraska, Tim; Demiralp, Çağatay; Hidalgo, César; Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In: SIGKDD. KDD '19, ACM, New York, NY, USA, p. 1500–1508, 2019.
- [Mi16] Mitlöhner, Johann; Neumaier, Sebastian; Umbrich, Jürgen; Polleres, Axel; Characteristics of Open Data CSV Files. In (IEEE, ed.): International Conference on Open and Big Data (OBD). IEEE, pp. 72 – 79, 2016.
- [NUP16] Neumaier, Sebastian; Umbrich, Jürgen; Polleres, Axel; Automated Quality Assessment of Metadata across Open Data Portals. *J. Data and Information Quality*, 8(1), oct 2016.
- [OP21] Oliveira, Daniela; Pesquita, Catia; , SemTab 2021 BioTable Dataset, October 2021. <https://doi.org/10.5281/zenodo.5606585>.
- [Pl18] Plotly; , Plotly. <https://chart-studio.plotly.com/feed/>, 2018.
- [Sh48] Shannon, Claude Elwood; A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 1948.
- [Su22] Suhara, Yoshihiko; Li, Jinfeng; Li, Yuliang; Zhang, Dan; Demiralp, Çağatay; Chen, Chen; Tan, Wang-Chiew; Annotating Columns with Pre-Trained Language Models. In: SIGMOD. ACM, New York, NY, USA, pp. 1493—1503, 2022.
- [Vi07] Viegas, Fernanda B.; Wattenberg, Martin; van Ham, Frank; Kriss, Jesse; McKeon, Matt; ManyEyes: A Site for Visualization at Internet Scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, nov 2007.
- [Zh20] Zhang, Dan; Hulsebos, Madelon; Suhara, Yoshihiko; Demiralp, Çağatay; Li, Jinfeng; Tan, Wang-Chiew; Sato: Contextual Semantic Type Detection in Tables. In: VLDB. volume 13. VLDB Endowment, p. 1835–1848, jul 2020.

# Reliable Rules for Relation Extraction in a Multimodal Setting

Björn Engelmann,<sup>1</sup> Philipp Schaer<sup>2</sup>

**Abstract:** Relation extraction for automated knowledge base construction typically requires much training data. If these are not available for a specific information need, relations must be extracted manually, or by hand-crafted extraction rules [Wu18]. Data Programming can be used to define heuristics that generate noisy labels for many instances, but this requires programming knowledge [Di19]. We present an approach to extract relations from multimodal documents using a few training data. Furthermore, we derive explanations in the form of extraction rules from the underlying model to ensure the reliability of the extraction. Finally, we will evaluate how reliable (high model fidelity) extracted rules are and which type of classifier is suitable in terms of F1 Score and explainability.

**Keywords:** Relation Extraction; Knowledge Extraction; Knowledge Base Construction; Explainable AI; Multimodal Documents

## 1 Introduction

*Automatic knowledge base construction* is a task that typically requires a large amount of labelled data against which extraction models can be trained. Unfortunately, especially in relation extraction, these labels are often unavailable since concrete use cases frequently differ strongly from each other. The corpus documents vary regarding the language used, data modality, structuredness, and domain. Many documents are also multimodal, which means that in addition to the text, they contain much other information, such as tables, font size, and text alignment. For this reason, annotated relations often must first be created in a laborious and error-prone procedure [Wu18].

An alternative to manual annotation is the application of hand-crafted extraction rules, which can automatically create noisy labels for many data instances [Ra17]. However, creating these rules requires programming knowledge, and without gold data, it is impossible to evaluate whether the applied rules are reliable. Furthermore, the application of complex language models in the low-resource setting is often infeasible due to the high computational effort involved [Ga21].

Due to these challenges, we present an approach that finds reliable extraction rules based on a small number of annotations, which makes the extraction model explainable on the

---

<sup>1</sup> Technische Hochschule Köln, Information Retrieval Research Group [bjoern.engelmann@th-koeln.de](mailto:bjoern.engelmann@th-koeln.de)

<sup>2</sup> Technische Hochschule Köln, Information Retrieval Research Group [philipp.schaer@th-koeln.de](mailto:philipp.schaer@th-koeln.de)

one hand and suitable to annotate new relations on the other hand. This has the advantage that a user can assess the model's reliability without having a lot of test data available. A small amount of training data is called a number less or equal to 10 instances in our context of information extraction. We follow the convention of few-shot learning, where few-shot means that only a few labelled training instances are available [De22]. These extraction rules are presented to a user, who can decide by expert feedback whether a rule fits their use case. The requirements for such a user are generally the necessary domain knowledge about the relations of the use case and basic HTML knowledge. A typical user group for this are data journalists, who often have HTML knowledge but not necessarily programming knowledge.

This work presents an approach that allows the integration of expert knowledge and expands the group of potential users for extracting relations in a low-resource setting without much labeling or programming effort. To ensure this, our model requires only a small amount of training data and provides extraction rules with high fidelity, which are suitable for user-driven feedback. For this purpose, we combine approaches from Explainable AI with those from Data Programming. If these rules are considered reliable by a user, it can reduce the annotation process for a new data science project.

The remainder of this paper is structured as follows: section 2 describes relevant related work. Then, in section 3, we introduce our overall approach using the associated pipeline and the methodological details. Next, our evaluation, experimental settings and the dataset used are presented in section 4. Finally, section 5 discusses the results and shows future work.

## 2 Related Work

The construction of a Knowledge Base is challenging, as Knowledge Bases need to be accurate, up-to-date, comprehensive, flexible, and efficient as possible. [Di19] propose automated knowledge base construction requirements that are not fully covered by any of the systems studied. An important key feature is an option for user feedback in which they can define or select extraction rules without coding skills.

Fonduer [Wu18] is a tool that implements a complete pipeline for the extraction of relations. It is based on the fact that users define the extraction rules themselves and evaluate and constantly improve them in an iterative process. Documents are automatically parsed, and their multimodal information, such as the membership of a span to a table header, is preserved. The user-defined extraction rules thus form entity types and relation types. However, these rules require programming knowledge, and the final extraction model can only be explained indirectly based on the defined rules.

By 2003, several methods for adaptive information extraction had already been presented. These focused roughly on two approaches. On the one hand, knowledge extraction with the

help of finite state techniques which are expressed by grammars or automata. On the other hand, relational rule learning techniques, where rules are learned in a Prolog style [KT03].

Modern approaches based on language models can consider the context of entities (neighbourhood of elements in the DOM tree) in HTML documents for extraction. In a few-shot setting, attributes can be extracted from a web page by pre-training the model on unlabeled web pages [De22]. In this way, an average of 10 training websites is sufficient to achieve an attribute value-level F1 score of 94.2 for an attribute extraction task on a website.

[Ha17] have presented a tool that provides a user with a visual interface to perform simple information extraction tasks without knowing a programming language. Easy to understand extraction rules are presented, which are generated from a small set of labelled data. Their system already has a bunch of predefined rules. These rules can then be refined to improve extraction performance. However, it is impossible to relate these rules to a trained model, which makes it impossible to perform more complex extraction tasks.

Explainable Artificial Intelligence is a research field that aims to make AI systems results more understandable to humans [AB18]. Approaches to make the behaviour of these black-box systems understandable are, e.g., Rule Extraction or Feature Importance. Lime [RSG16a] can be used to generate explanations in the form of feature importance scores within a local neighbourhood around the instance to be explained. To do this, new artificial instances are sampled near the input to be explained, which are then used to learn a simple local regression model. The learned coefficients then correspond to the feature scores. To evaluate the comprehensibility of extracted explanations, [Ji21] conducted a survey and assessed which types of explanations are well suited to improve the understanding of a black-box model. Users found extracted rules very helpful, especially when they refer to a few features.

To link user feedback with Explainable Artificial Intelligence, [TK18] give a user the possibility to mark a feature for a given training example in such a way that this feature should not influence the training process. The user can determine whether the model has mistakenly drawn a connection between this feature and the example through domain knowledge.

The advantage of our approach is that arbitrary extraction models can be used, from which explanations can be extracted using Explainable AI. This is not provided for in the classical techniques of adaptive learning. In principle, our approach is compatible with all black-box classifiers and all XAI methods that provide extraction rules. Prolog-like rule systems, for example, do not offer this flexibility since, at most, the rule systems themselves can serve as explanations.

To our knowledge, no system has been presented that allows users without programming knowledge to extract reliable relations from multimodal data based on a few training examples.

### 3 Methods

This section describes the individual building blocks of our approach. In Figure 1, the procedure is roughly shown.

A knowledge base construction framework (Fonduer [Wu18]) parses multimodal documents, and relation candidates are generated. The user labels a small set of documents (subsection 3.1).

Relations go through the steps of featurization, dimension reduction and feature combination (subsection 3.2, subsection 3.3).

The transformed vectors are used to train the model. The model provides predictions and feedback to the user through a ranked list of explanations (subsection 3.4).

The user gives feedback in the form of a selection of reliable rules (subsection 3.5).

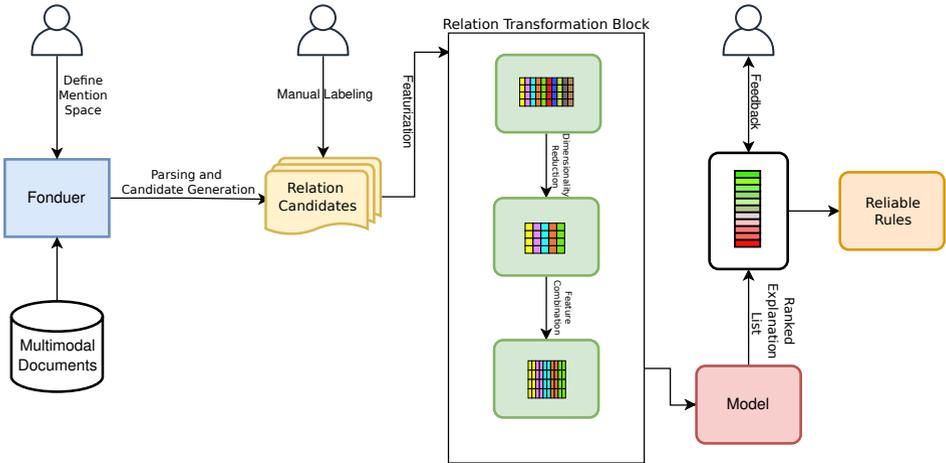


Fig. 1: Process of the overall approach in a productive environment. In the relation transformation block, rows represent instances, columns express individual entity features, and colours symbolise a component in the feature vector.

#### 3.1 Parsing and Candidate Generation

Parsing multimodal documents is challenging because different kinds of information (font, table structure, colour) should be preserved. However, as much information as possible should be available in the database in a uniform and structured way. For this purpose, the framework Fonduer is used in this work [Wu18]. After parsing, a hierarchically structured graph is available for each document. For example, a section can contain text, tables, or

figures. The smallest unit consists of sentences. Fonduer captures the context of sentences and candidates, e.g. information about where they appear in the document. Furthermore, details such as font size or the HTML class used for an element are preserved.

This form of modelling makes it possible to use the structural information of the document as a signal for relation extraction. In our case, relation candidates consist of two mentions, which are two text spans in the document that potentially express the user’s desired relation. Both correct and incorrect relation candidates are required to train the model to solve the binary classification problem. Correct relations are those candidates where both entity mentions are in a predefined connection. Incorrect relations contain entity mentions that are randomly drawn from the document. Accordingly, all relation candidates are derived from the cartesian product of both mention sets. Furthermore, we make sure that some incorrect candidates contain exactly one correct entity mention (details in subsection 4.2). The set of correct relation candidates is manually assigned. Fonduer then transforms these candidates into a feature space that embeds textual, structural, visual, and tabular features. We denote the transformed correct candidates as  $\mathcal{R}_{pos}$  and the incorrect  $\mathcal{R}_{neg}$ , respectively.

### 3.2 Featurization and Dimensionality Reduction

With the Fonduer featurization, each relation candidate is assigned to a feature vector of dimension  $D$  derived from the multimodal context of the linked entities. Each feature is binary and expresses whether a property applies to a relation or not (e.g., the first entity mention contains the word *professional*, some instances can be seen in Table 2). This feature vector is denoted as  $\mathbf{r} = \{r_1, \dots, r_D\}$ . Each component  $r_i$  of the feature vector refers to a property of the respective entity mention of entity type  $e_0$  or  $e_1$ . We denote the set of all properties of an entity type  $\mathcal{E}_0$  and  $\mathcal{E}_1$ , respectively. Furthermore, some properties refer to the context of both entity mentions (e.g., the distance of both mentions from each other), the total set of which we call  $\mathcal{G}$ . We denote the set of all features  $\mathcal{F} = \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{G}$ .

The dimensionality of a feature vector is typically over 100k for a common set of HTML documents. However, since we have little training data available and want to make model decisions explainable, we reduce the dimensionality [Cu08]. This is achieved by filtering out the vector components that vary least from the difference between the average correct and incorrect training vectors. We use this form of dimension reduction to obtain binary features. Since the explainability of single features should be preserved, we cannot use techniques like singular value decomposition or embeddings because we would obtain a real feature space. Our dimension reduction approach assumes that those features are particularly relevant for the classification whose values differ most between the positive and negative candidates. The  $l$  most varying components are defined in the following way:

$$\mathcal{F}_{sig} = \text{argsort}_l (|\overline{\mathbf{r}_{pos}} - \overline{\mathbf{r}_{neg}}|). \quad (1)$$

$$\overline{\mathbf{r}}_{pos} = \frac{1}{|\mathcal{R}_{pos}|} \sum_{\mathbf{r}^{(i)} \in \mathcal{R}_{pos}} \mathbf{r}^{(i)}. \quad (2)$$

The  $\text{argsort}_l$  function returns indices of the  $l$  vector components with the highest value. The average of all vectors corresponding to the incorrect relations  $\overline{\mathbf{r}}_{neg}$  is defined respectively. Thus,  $\mathcal{F}_{sig}$  contains the  $l$  indices of such components that differ most with respect to correct and incorrect relations. We assume that the corresponding features are the most important for our relation classification task and discard the rest.

### 3.3 Feature Combination

Another approach we take is to combine features from  $\mathcal{E}_0$  and  $\mathcal{E}_1$  into one feature. As explained in subsection 3.1, there are incorrect relations where one of two entity mentions is correct. This is because each relation, in our case, consists of exactly two entities. Since individual features are intended to serve as both a classification explanation and an extraction rule, it is reasonable to require the validity of two relation properties in one rule. Therefore, explanations and extraction rules should also express both properties in one. Here we use only those features that are preserved after dimension reduction. We denote the set of all combinations of  $\mathcal{E}_0$  and  $\mathcal{E}_1$  together with the features from  $\mathcal{G}$ ,  $\mathcal{F}_{combined}$ , where  $\mathcal{F}_{combined} = (\mathcal{E}_0 \times \mathcal{E}_1) \cup \mathcal{G}$ .

### 3.4 Training and Explanations

For the binary classification task, we use low-complexity models (details in subsection 4.2) because their training is better suited in the low-resource setting, and there is evidence that the complexity of models correlates negatively with their explainability [Gu19].

After the training, we extract a set of explanations with Lime. Lime is an approach that explains the prediction of a specific instance based on the importance weights of the associated features [RSG16b]. Lime explains a selected instance, but since we want to obtain extraction rules that explain the overall model and provide valuable explanations, we need to choose a representative but also a diverse set of instances for Lime. We have chosen this local approach because the dimension reduction already performs a global selection, and the local approach results in a multitude of explanations. In addition, this has the advantage that a user can select one of these explanations. We generate diversified artificial instances based on our test relations to obtain explanations that cover as many relation patterns as possible. We build clusters over our test vectors using the k-means procedure to achieve this. The rounded cluster centres then form our representative, diverse instances based on which explanations are extracted. We derive a ranked list of feature combinations by summarizing the weights over all instances and sorting them in descending order. The intuition is that

each feature can be interpreted as a rule to classify unseen data. The higher the explanation weight of a rule is, the closer its predictions are to the predictions of the explained model.

Another approach to measuring the fidelity of each feature to the model is to apply each feature, interpreted as a rule, to the test data and then compare the results to the model predictions. Thus, a baseline is established that assigns an F1 Fidelity to each rule, which is derived from the F1 Score of the model predictions and the application of each rule [Gu18a].

### 3.5 User Feedback

The ranked explanation list can then be presented to a user who selects a reliable rule based on domain knowledge to classify relation candidates. Under the assumption that the list position correlates with the actual F1 Score, the advantage is that a user has less effort in selecting a reliable rule. In subsection 4.4, an example of a ranked list is shown, in addition to the quantitative analysis, to make plausible that some rules are both understandable and accurate. We present rules that we assume a user would plausibly choose to find an appropriate expression based on the context of the use case.

## 4 Experiments

The following subsections evaluate which classifiers are suitable for relation candidate classification and generating reliable extraction rules based on different amounts of training data. The test data labels are only used to evaluate the final results. We never use the test labels for dimension reduction or hyperparameter selection. Our code and data are available at [https://osf.io/dn9hm/?view\\_only=7e65fd1d4aae44e1802bb5ddd3465e08](https://osf.io/dn9hm/?view_only=7e65fd1d4aae44e1802bb5ddd3465e08).

### 4.1 Dataset

The Structured Web Data Extraction (SWDE) dataset consists of a collection of 124,291 structured web pages with 8 different verticals [Hal1]. A vertical (e.g., job posting) consists of 10 differently formatted websites, each consisting of up to 2000 pages. Within a page are labelled attributes (in the case of job postings: title, company, location, date). For our case, we want to extract relations between job titles and corresponding locations. All websites are available in HTML. Since there can be many mentions of job titles and locations on each website, only those relation mentions are considered correct whose entity mentions are at the correct position in the document. An example of a job posting can be seen in Figure 2. We use 400 webpages from the Careerbuilder site. We define the mention space for all jobtitle entity mentions as n-grams between 1-9 items and 1-6 items for the location mentions, respectively.

|  |  |
|--|--|
| <p>Base Pay: \$80,000 - \$120,000 /Year</p> <p>Other Pay:</p> <p>Employee Type: Full-Time</p> <p>Industry: Computer<br/>Software<br/>Computer<br/>Hardware<br/>Banking -<br/>Financial<br/>Services</p> <p>Manages No<br/>Others:</p> <p>Job Type: Information<br/>Technology<br/>Finance<br/>Banking</p> <p>Required Education: 4 Year Degree</p> <p>Required Experience: At least 5 year(s)</p> <p>Travel: Required Not Specified</p> <p>Relocation Covered: Not Specified</p> <p>Reference ID: Not Available</p> <p>Location:  Chicago</p> <p> Loading Map...</p> <p>Contact: Alex Purvis<br/>Phone: Not Available<br/>Email: <a href="#">Send Email Now</a><br/>Fax: Not Available</p> | <p><b>SQL Reports Developer</b> <a href="#">Apply Now</a><br/><a href="#">Report It</a></p> <hr/> <p><b>JOB DESCRIPTION</b></p> <p>Our client is a large professional services firm located in Chicago in need of a senior reports / database developer. This developer would need to have a strong business analyst background and would be responsible for developing customized reports and other database development in SQL server 2005. This position is responsible for leading the support of the design, implementation, and maintenance of database solutions; Front-Office Financial Report Development; supporting end-to-end report development, management, and delivery of these solutions; interfacing between various business and technical teams to compile requirements and design solutions.</p> <hr/> <p><b>JOB REQUIREMENTS</b></p> <p>Required skills:</p> <ul style="list-style-type: none"> <li>-5+ years of SQL Development in the financial industry</li> <li>-5+ years experience with SQL 2000/2005</li> <li>-5+ years of reports development including SSRS, and Crystal Reports</li> <li>-SQL database experience should include stored procedures, functions, triggers, developing views, performance tuning, query optimization</li> </ul> <p>-C# development experience is a plus</p> |
|--|--|

Fig. 2: Example excerpt of a job posting from the Careerbuilder website. The entities of the correct relation are marked in green.

## 4.2 Experimental Settings

For the experiments, different types of models are used to investigate the relationship between classification performance and explainability. For all of the following models, the sklearn default configurations are used: Multi-Layer Perceptron (MLP), Decision Tree (DT), k-Nearest Neighbors (kNN), Gradient Boosting (GB), Random Forest (RF), Support Vector Classifier (SVM), Naive Bayes (NB) [Pe11]. Since we want to evaluate our approach for a small number of training data, we limit the number of correct relations  $|\mathcal{R}_{pos}|$  and test the following amounts:  $|\mathcal{R}_{pos}| \in \{3, 5, 10, 20, 40\}$ . We sample 10 incorrect relations for each correct relation, which is a typical ratio for relation extraction [NG15]. Under these 10 incorrect relations are two containing exactly one correct entity span. Since we know that a document can contain only one correct relation, the remaining combinations of mentions can serve as the basis of the incorrect relations. We use this variety of simple classifiers to evaluate whether differences in explainability can be detected. We also use the standard deviation of classification performance over multiple training runs to assess how reliable a model is for a given set of training data. The larger the standard deviation of the F1 score of a model, the less reliable the extraction performance.

### 4.3 Ablations

We evaluate each module of our pipeline in terms of median F1 Score. Thus, each model type is evaluated with the totality of all features, with the features after dimensionality reduction, and with the combined features. Each configuration is evaluated with 10-Fold cross-validation to determine standard deviation and median values. The scatter values of the classifiers are particularly important since this is an indicator of model reliability. We use the scatter of the F1 Score to measure the model’s reliability, as we don’t know the actual F1 Score in a productive setting where we have no labelled test data. Especially when little training data is used, a higher scatter of F1 Score results (as seen in Table 1).

For k-means clustering, we use 10 cluster centres, and dimension reduction reduces the feature space from 382k dimensions to 30. Dimension reduction almost always resulted in better F1 Scores. This can be seen particularly clearly for Naive Bayes and SVM. When applying the feature combination, no clear pattern emerges; only the standard deviation for the Random Forest decreases and a constantly increased F1 Score for Naive Bayes. It is also noticeable that for the Random Forest model with combined features in the median already, 5 correct training relations are sufficient to achieve an F1 Score of 1.0, with a standard deviation of 0.1.

Tab. 1: Median F1 Scores and corresponding standard deviations for different training amounts and model types.

| Model / # train | 2         | 3         | 5               | 10        | 20        | 40        |
|-----------------|-----------|-----------|-----------------|-----------|-----------|-----------|
| MLP full        | 0.84±0.06 | 0.85±0.03 | 0.87±0.01       | 0.9±0.02  | 0.94±0.02 | 0.95±0.02 |
| MLP red.        | 0.96±0.08 | 0.98±0.06 | 0.99±0.07       | 1.0±0.02  | 1.0±0.01  | 1.0±0.0   |
| MLP comb.       | 0.96±0.04 | 0.97±0.02 | 0.98±0.02       | 0.99±0.01 | 1.0±0.01  | 1.0±0.0   |
| DT full         | 0.86±0.12 | 0.79±0.1  | 0.85±0.06       | 0.9±0.04  | 0.94±0.03 | 0.95±0.01 |
| DT red.         | 0.86±0.12 | 0.85±0.06 | 0.92±0.08       | 0.94±0.05 | 0.96±0.03 | 1.0±0.01  |
| DT comb.        | 0.93±0.05 | 0.91±0.07 | 0.97±0.03       | 1.0±0.04  | 1.0±0.01  | 1.0±0.0   |
| KNN full        | 0.2±0.32  | 0.29±0.15 | 0.42±0.12       | 0.57±0.11 | 0.63±0.07 | 0.72±0.07 |
| KNN red.        | 0.97±0.01 | 0.98±0.02 | 0.99±0.01       | 1.0±0.01  | 1.0±0.01  | 1.0±0.0   |
| KNN comb.       | 0.97±0.06 | 0.98±0.01 | 0.99±0.01       | 1.0±0.01  | 1.0±0.01  | 1.0±0.01  |
| RF full         | 0.93±0.27 | 0.97±0.04 | 0.94±0.03       | 1.0±0.02  | 1.0±0.01  | 1.0±0.0   |
| RF red.         | 0.93±0.09 | 0.96±0.09 | 0.98±0.08       | 1.0±0.01  | 1.0±0.01  | 1.0±0.0   |
| RF comb.        | 0.98±0.01 | 0.98±0.02 | <b>1.0±0.01</b> | 1.0±0.01  | 1.0±0.01  | 1.0±0.0   |
| GB full         | 0.86±0.04 | 0.85±0.06 | 0.93±0.07       | 0.94±0.05 | 0.96±0.04 | 0.97±0.01 |
| GB red.         | 0.86±0.06 | 0.85±0.07 | 0.95±0.08       | 0.96±0.05 | 0.95±0.03 | 1.0±0.01  |
| GB comb.        | 0.97±0.01 | 0.98±0.02 | 0.98±0.01       | 0.99±0.01 | 1.0±0.02  | 1.0±0.0   |
| NB full         | 0.01±0.24 | 0.07±0.27 | 0.64±0.26       | 0.64±0.3  | 0.96±0.02 | 0.92±0.03 |
| NB red.         | 0.74±0.17 | 0.76±0.12 | 0.75±0.02       | 0.78±0.06 | 0.77±0.04 | 0.77±0.01 |
| NB comb.        | 0.9±0.27  | 0.89±0.06 | 0.94±0.03       | 0.92±0.03 | 0.93±0.03 | 0.96±0.02 |
| SVM full        | 0.0±0.0   | 0.0±0.0   | 0.0±0.0         | 0.0±0.0   | 0.0±0.0   | 0.0±0.0   |
| SVM red.        | 0.93±0.17 | 0.96±0.03 | 0.97±0.03       | 1.0±0.03  | 1.0±0.02  | 1.0±0.0   |
| SVM comb.       | 0.87±0.17 | 0.9±0.07  | 0.92±0.07       | 0.98±0.05 | 0.99±0.02 | 1.0±0.0   |

#### 4.4 Explanation Evaluation

Tab. 2: Selection of the 10 extracted rules with the highest F1 Fidelity.

| F1 Fidelity Explanation  | F1 Score |
|--|----------|
| STR_e0_HTML_ATTR_class=job_title AND STR_e1_NEXT_SIB_TAG_iframe        | 1.0      |
| STR_e0_HTML_ATTR_class=job_title AND BASIC_e1_CONTAINS_WORDS_[US]      | 1.0      |
| STR_e0_HTML_ATTR_class=job_title AND STR_e1_HTML_ATTR_rel=nofollow     | 1.0      |
| STR_e0_HTML_ATTR_class=job_title AND BASIC_e1_CONTAINS_WORDS_[US -]    | 1.0      |
| STR_e0_HTML_ATTR_class=job_title AND STR_e1_HTML_ATTR_class=BingMap    | 1.0      |
| STR_e0_HTML_ATTR_class=job_title AND STR_e1_HTML_ATTR_id=JobDetails_.. | 1.0      |
| STR_e0_ANCESTOR_TAG_[html body ...] AND STR_e1_NEXT_SIB_TAG_iframe']   | 0.98     |
| STR_e0_ANCESTOR_TAG_[html body ...] AND BASIC_e1_CONTAINS_WORDS_[US]   | 0.98     |
| STR_e0_ANCESTOR_TAG_[html body ...] AND STR_e1_HTML_ATTR_rel=nofollow' | 0.98     |
| STR_e0_ANCESTOR_TAG_[html body ...] AND STR_e1_HTML_ATTR_class=BingMap | 0.98     |

Since our goal is to extract reliable rules, we evaluate the fidelity of the explanations of all model types using the F1 Fidelity between explanation and prediction [Gu18b]. To evaluate the quality of the final ranking, we calculate the rank correlation between an optimal ranked list (according to the F1 Score for a specific rule against the test labels) and a list resulting from ordering the explanation weight. The set of rules to be ordered is the same here, only the order may differ. We used the Spearman rank correlation instead of the Pearson correlation coefficient since the values of the explanation weights are no longer relevant for the ranking, only their order. Furthermore, the distribution of the explanation weights does not necessarily follow a normal distribution, which must be assumed for the Pearson correlation coefficient.

Figure 3 illustrates how the number of training data, the model, and the explanation type affect the rank correlation. Extracted rules ordered by F1 Fidelity correlate more strongly with an optimally ranked list than a list ordered by Lime explanation weights. Furthermore, it is shown that RF and KNN achieve a rank correlation of more than 0.98 from 5 correct training relations. The Lime explanation weights for SVM and DT were omitted because they do not have a function to assign pseudo-probabilities to instances. In general, the rank correlation tends to improve for an increasing number of training data.

In Table 2, the top ten explanations extracted from a Gradient Boost model are shown as an

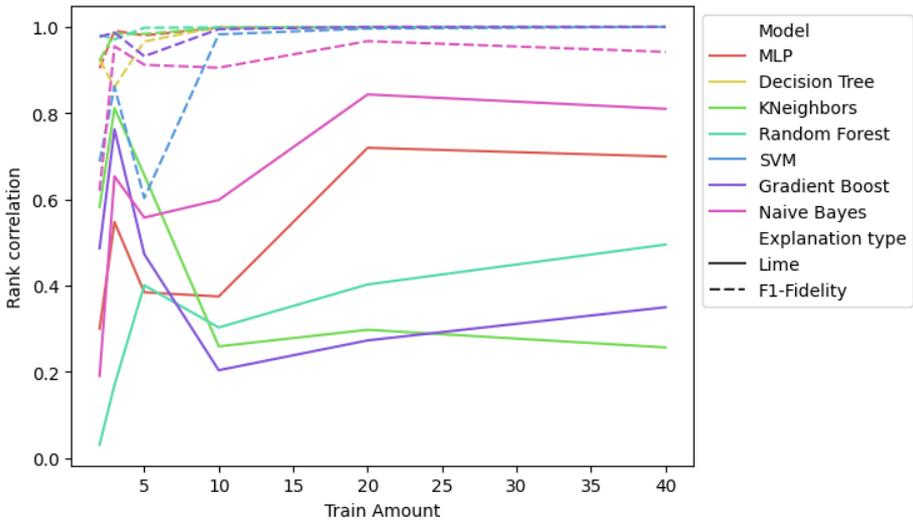


Fig. 3: Rank correlations for different models and explanation types plotted against the number of correct training relations. Only for models with combined features.

example. This was trained with 5 correct relations. According to Table 1, this configuration has an F1 Score of 0.98, while all top six extracted rules have an F1 Score of 1.0. We assume that a user would select rule #5 as reliable. Based on the HTML classes, the user can infer the meaning of the entities because the *jobtitle class* indicates a correct jobtitle mention, and the *BingMap class* expresses the presence of a corresponding location (Figure 2). The authors from [De22] use a complex language model to achieve an attribute value-level F1 score of 94.8 for a similar task. The results cannot be compared directly because their model does not use candidate generation; therefore, it is not a classification task but an attribute extraction task. Also, the model is trained on 80 different sites and thus has to recognize a larger variety of patterns. However, an average number of 10 webpages was used for the few-shot training.

## 5 Discussion

In this work, we presented an approach to extract relations and corresponding rules from multimodal documents using a small amount of training data. Using our example from Table 2, it can be seen that even a single rule can provide better extraction performance than the underlying model. The prerequisite for this is that a user would select this rule.

In this way, annotating new websites with less labelling effort is possible. This is the case because the user would have to use part of the annotated data in a setting without explanations to evaluate the extraction model. However, more than 5 annotated websites

would be necessary for a reliable evaluation. Reliable rules can then be used to annotate unknown data.

Rules extracted by Lime perform worse than those extracted by the baseline method. We assume this is because Lime is unsuitable for classification problems where many features provide a strong signal for the correct class. The main area for improvement in this work is the simplicity of the data set and the associated extraction task. Future work is to apply the presented approach to more complex data. Furthermore, more advanced approaches to explanatory extraction, such as Lore [Gu18c], will be used.

## Bibliography

- [AB18] Adadi, Amina; Berrada, Mohammed: Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.
- [Cu08] Cunningham, Pádraig: Dimension Reduction. In (Cord, Matthieu; Cunningham, Pádraig, eds): *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 91–112, 2008.
- [De22] Deng, Xiang; Shiralkar, Prashant; Lockard, Colin; Huang, Binxuan; Sun, Huan: , DOM-LM: Learning Generalizable Representations for HTML Documents, 2022.
- [Di19] Din, Osman: Towards a Flexible System Architecture for Automated Knowledge Base Construction Frameworks. In: 2019 IEEE International Conference on Big Data (Big Data). pp. 3066–3071, 2019.
- [Ga21] Ganesh, Prakhar; Chen, Yao; Lou, Xin; Khan, Mohammad Ali; Yang, Yin; Sajjad, Hassan; Nakov, Preslav; Chen, Deming; Winslett, Marianne: Compressing Large-Scale Transformer-Based Models: A Case Study on BERT. *Transactions of the Association for Computational Linguistics*, 9:1061–1080, 09 2021.
- [Gu18a] Guidotti, Riccardo; Monreale, Anna; Ruggieri, Salvatore; Pedreschi, Dino; Turini, Franco; Giannotti, Fosca: , Local Rule-Based Explanations of Black Box Decision Systems, 2018.
- [Gu18b] Guidotti, Riccardo; Monreale, Anna; Ruggieri, Salvatore; Pedreschi, Dino; Turini, Franco; Giannotti, Fosca: , Local Rule-Based Explanations of Black Box Decision Systems, 2018.
- [Gu18c] Guidotti, Riccardo; Monreale, Anna; Ruggieri, Salvatore; Pedreschi, Dino; Turini, Franco; Giannotti, Fosca: Local Rule-Based Explanations of Black Box Decision Systems. *CoRR*, abs/1805.10820, 2018.
- [Gu19] Gunning, David; Stefik, Mark; Choi, Jaesik; Miller, Timothy; Stumpf, Simone; Yang, Guang-Zhong: XAI&#x2014;Explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120, 2019.
- [Ha11] Hao, Qiang: Structured Web Data Extraction Dataset (SWDE). 2011.
- [Ha17] Hanafi, Maeda F.; Abouzied, Azza; Chiticariu, Laura; Li, Yunyao: SEER: Auto-Generating Information Extraction Rules from User-Specified Examples. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, p. 6672–6682, 2017.

- [Ji21] Jin, Weina; Fan, Jianyu; Gromala, Diane; Pasquier, Philippe; Hamarneh, Ghassan: EUCA: A Practical Prototyping Framework towards End-User-Centered Explainable Artificial Intelligence. CoRR, abs/2102.02437, 2021.
- [KT03] Kushmerick, Nicholas; Thomas, Bernd: Adaptive Information Extraction: Core Technologies for Information Agents. In (Klusch, Matthias; Bergamaschi, Sonia; Edwards, Pete; Petta, Paolo, eds): Intelligent Information Agents: The AgentLink Perspective. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 79–103, 2003.
- [NG15] Nguyen, Thien Huu; Grishman, Ralph: Relation extraction: Perspective from convolutional neural networks. In: Proceedings of the 1st workshop on vector space modeling for natural language processing. pp. 39–48, 2015.
- [Pe11] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.
- [Ra17] Ratner, Alexander; Bach, Stephen H.; Ehrenberg, Henry R.; Fries, Jason Alan; Wu, Sen; Ré, Christopher: Snorkel: Rapid Training Data Creation with Weak Supervision. CoRR, abs/1711.10160, 2017.
- [RSG16a] Ribeiro, Marco Túlio; Singh, Sameer; Guestrin, Carlos: "Why Should I Trust You?": Explaining the Predictions of Any Classifier. CoRR, abs/1602.04938, 2016.
- [RSG16b] Ribeiro, Marco Tulio; Singh, Sameer; Guestrin, Carlos: "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. pp. 1135–1144, 2016.
- [TK18] Teso, Stefano; Kersting, Kristian: , "Why Should I Trust Interactive Learners? Explaining Interactive Queries of Classifiers to Users, 2018.
- [Wu18] Wu, Sen; Hsiao, Luke; Cheng, Xiao; Hancock, Braden; Rekatsinas, Theodoros; Levis, Philip; Ré, Christopher: Fondue: Knowledge Base Construction from Richly Formatted Data. In: Proceedings of the 2018 International Conference on Management of Data. SIGMOD '18, Association for Computing Machinery, New York, NY, USA, p. 1301–1316, 2018.



# Predictive Maintenance for the Optical Synchronization System of the European XFEL: A Systematic Literature Survey

Arne Grünhagen<sup>1,2,3,4</sup>, Marina Tropmann-Frick<sup>1</sup>, Annika Eichler<sup>2,3</sup>, Görschwin Fey<sup>2</sup>

**Abstract:** The optical synchronization system of the European X-ray Free Electron Laser is a networked cyber-physical system producing a large amount of data. To maximize the availability of the optical synchronization system, we are developing a predictive maintenance module that can evaluate and predict the condition of the system. In this paper, we report on state-of-the-art predictive maintenance methods by systematically reviewing publications in this field. Guided by three research questions addressing the type of cyber-physical systems, feature extraction methods, and data analytical approaches to evaluate the current health status or to predict future system behavior, we identified 144 publications of high quality contributing to research in this area. Our result is that especially neural networks are used for many predictive maintenance tasks. This review serves as a starting point for a detailed and systematic evaluation of the different methods applied to the optical synchronization system.

**Keywords:** Predictive maintenance; Condition monitoring; Fault analysis; Cyber-physical systems; Systematic literature review

## 1 Introduction

The European X-ray Free Electron Laser (EuXFEL) is the largest currently operated linear particle accelerator in the world and opens cutting-edge research opportunities in molecular and material science and system biology on atomic scale [So20]. Those precise measurements require timing with an error margin in the femtosecond range for most subsystems within the facility. To provide this high-precision timing, an optical synchronization system is installed at the facility to synchronize critical accelerator components in time. Due to the high demands on operating the optical synchronization system accurately, even small decreases in performance can have a huge impact on the overall system [Sc19].

To monitor the health status of the optical synchronization system, different kinds of sensors are installed for measuring environmental conditions like temperature or relative humidity, but also for monitoring more complex properties like numerous control loop variables.

---

<sup>1</sup> Hamburg University of Applied Sciences, HAW, Germany [firstname.secondname@haw-hamburg.de](mailto:firstname.secondname@haw-hamburg.de)

<sup>2</sup> Hamburg University of Technology, TUHH, Germany [firstname.secondname@tuhh.de](mailto:firstname.secondname@tuhh.de)

<sup>3</sup> Deutsches Elektronen-Synchrotron DESY, Germany [firstname.secondname@desy.de](mailto:firstname.secondname@desy.de)

<sup>4</sup> We acknowledge the support by DASHH (Data Science in Hamburg - HELMHOLTZ Graduate School for the Structure of Matter) with the Grant-No. HIDSS-0002.

Especially, the frequency domain of these control loop signals provides information about electrical, mechanical, and optical disturbances. Since the optical synchronization system contains several interconnected devices like laser oscillators, controllers, and motors, we consider the system as a networked cyber-physical system (CPS). Due to the huge complexity of the optical synchronization system and the partially high data rate (up to 300 kHz), detecting and tracking all kinds of failures is not feasible for a human. Therefore, we plan to develop an automated mechanism for the optical synchronization system that identifies and, if possible, prevents potential failures to decrease machine downtime.

The process of automatically identifying faulty behavior of a system and if possible initiating countermeasures using data-driven methods is known as Condition Monitoring (CM) [Ha11]. CM methods use signal processing techniques and fault analysis tools for evaluating the overall health of a system. Predictive Maintenance (PM) techniques try to predict future critical system conditions in advance to initiate countermeasures before potential bad system states occur [PVB21].

To get a full overview of what kind of PM and CM methods exist and which of them can be applied to the optical synchronization system, we conducted a systematic literature review that aims to determine state-of-the-art CM and PM techniques applied to CPS.

The rest of this review paper is organized as follows: Section 2 discusses related publications. Section 3 describes our approach to conducting a systematic literature review, including three research questions that we want to answer. Section 4 reports the main findings of the systematic literature review. Finally, we end with a conclusion in Section 5.

## 2 Prior research

Most publications reviewing CM, PM, or fault analysis methods give an overview of methods with respect to their respective research area but do not differentiate between CM and PM, i.e., robotics [Hu21, ITK19, Ki18a, Le18, MTT21], rotating machinery [Dr21, NUS21a, NUS21b], energy management [AC20, RTJ21a], transportation systems [AH20, Zh21b], or wind turbines [De21, RTJ21b].

Literature surveys focussing on predictive maintenance tend to evaluate the methods with respect to their industrial and economical context, i.e., [HB21, Ji20a, AA21, Ar21, BCC21]. The authors of [Bo19, SYD11, Li21] each report on predictive maintenance algorithms and future trends in their respective application areas. Literature surveys addressing condition based monitoring also focus on the concrete methods used in their respective application area, i.e, offshore-wind turbines [BRK21, Ma20], rail transport systems [KM21], and hydroelectric plants [dSGC22]. Since anomaly detection is a very prominent way to detect bad systems states, we are also interested in literature reviews covering anomaly detection for CPS. In [AC17, AKI21, Na21a, Se22] the authors review state-of-the-art anomaly detection methods being applied to time series sensor data of different CPS domains.

In conclusion, existing literature reviews focus on publications adhering to a domain different from ours. Furthermore, most existing publications do not differentiate between condition monitoring and predictive maintenance. Therefore, we conducted a systematic literature

review for identifying state-of-the-art methods and techniques that can be used for CM or PM for CPS.

### 3 Methodology

A systematic literature review is a formal and well-structured approach to synthesize evidence and thus allow researchers to come to an understanding of the current status and current challenges of a specific research area. The methodology of our systematic literature review follows the guidelines as proposed by [KC07]. A systematic literature review consists of four consecutive steps, namely Identification, Screening, Eligibility, and Quality (see Figure 1). First, a set of primary publications was built in the identification step that is successively reduced in the following steps. The filtering in each step is based on respective criteria that are based on a set of research questions.

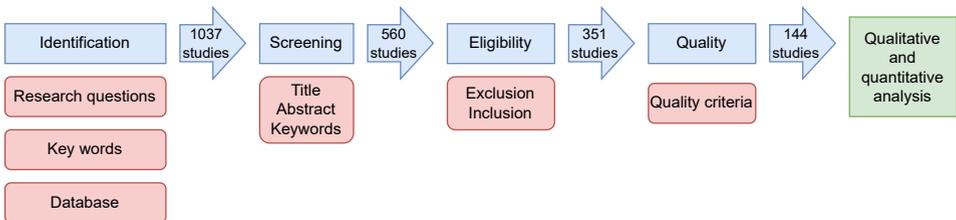


Fig. 1: Systematic literature process

#### 3.1 Research questions and contribution

Our systematic literature review is complementary to existing research in the field of CM and PM for CPS by addressing the research questions depicted in Table 1.

#### 3.2 Selection of primary studies

In the identification step, primary studies were identified by searching for specific keywords in well-known databases. The keywords are derived from the previously defined research questions (Table 1). The systematic literature review was carried out in July 2022 without any restrictions. The primarily identified studies originate from the databases **ACM Digital Library**, **IEEE Xplore**, **Elsevier Scopus**, **Springer Link**, and **Multidisciplinary Digital Publishing Institute**.

The keywords string is based on three different aspects, namely **CM and PM**, **data analysis**, and **Cyber-physical systems**. Each aspect is expanded with a list of various synonyms and phrases that have a similar meaning resulting in an aspect group. The keyword search string

Tab. 1: Research questions

| Research questions   | Discussion   |
|--|--|
| <b>RQ1:</b> What kind of data is used for monitoring and predicting the health of a CPS? | For analysing the health status of CPS it is required to access sensor data provided by that system.   |
| <b>RQ2:</b> What methods exist to extract meaningful features from data provided by CPS? | Sensor data can very often not directly be used for further machine learning tasks. Therefore the data recordings need some kind of data processing to make the data more meaningful. Thus, an overview of what feature extraction methods are commonly used in the literature as well as their respective CPS areas are determined. |
| <b>RQ3:</b> What methods exist for CM and PM for CPS?                                    | The most prominent data driven methods for PM and CM for CPS are identified.   |

is created by connecting the phrases and synonyms of each single aspect group with a logical OR and the three aspect groups are connected with a logical AND. We split the two aspect groups CM and PM and data analysis. Merging these two aspect groups, we came up with a lot of publications that do not follow our research goal which is to detect a degradation in the system. We also generalized the optical synchronization system part, where we found out that the term CPS is the most general form of a complex system like the optical synchronization system and is the best fit for our research goal. The final search string looks as follows:

```
('predictive maintenance' OR 'health monitoring' OR 'condition
monitoring') AND ('data analysis' OR 'fault diagnosis' OR 'fault
analysis' OR 'fault detection' OR 'anomaly detection' OR 'outlier
detection' OR 'time series forecasting' OR 'time series prediction'
OR 'data forecasting')
AND ('Cyber Physical System' OR 'CPS' OR 'Cyber-Physical System')
```

Combining the results of the different databases results in a total of 1037 studies.

### 3.3 Screening

In the initial screening phase, we filtered the studies following a set of very broad guidelines to ensure that no important studies are filtered out in the first stage. A publication passed the first screening phase if it follows one of the following criteria:

- The study describes what kind of data is extracted from a CPS (RQ1)
- The study presents how data coming from a CPS is processed (RQ2)
- The concept of CM or PM in the context of CPS is explained in general (RQ3)

- The study describes how a specific CM or PM method is considered in the context of CPS (RQ3)
- Different predictive maintenance methods are compared and evaluated (RQ3)

The number of studies was decreased by 477 to 560 remaining studies.

### 3.4 Eligibility and Evaluation

The eligibility of the remaining publications was determined by examining the full texts of the papers against a predefined set of inclusion and exclusion criteria (see Table 2).

Tab. 2: Inclusion and exclusion criteria for the studies

| Inclusion Criteria   | Exclusion Criteria   |
|--|--|
| <ul style="list-style-type: none"><li>• Original research study</li><li>• Peer-reviewed publication</li><li>• Study presents new methods for CM or PM for CPS</li><li>• Study evaluates CM or PM methods for CPS</li></ul> | <ul style="list-style-type: none"><li>• Secondary research and review papers</li><li>• Studies that are only available as presentations</li><li>• Publications not in English or German</li><li>• Studies covering network security of connected CPS</li></ul> |

To proceed to the next evaluation phase, a study has to meet three of the four inclusion criteria and none of the exclusion criteria. In this phase, we reduced the number of studies by 209 to 351 remaining studies.

### 3.5 Quality Assessment and Synthesis

Each of the remaining studies is evaluated using a set of quality assessment criteria depicted in Table 3. Each study gets assigned a score between 0 and 6. All studies with a score of less or equal to 3 are excluded. After the quality assessment phase, we have a total of 144 publications of high quality according to our guidelines.

## 4 Data analysis

To provide insights into the current state and future trends in CM and PM for CPS, we performed a descriptive analysis of the remaining publications attained through the systematic attrition process (see Fig 1). Afterward, we performed a detailed qualitative analysis of the selected literature, addressing each research question individually.

Tab. 3: Quality assessment parameters

| Parameter               | Quality indicator  | Score |
|-------------------------|--|-------|
| CPS environment         | No description of the CPS and the data used              | 0     |
|                         | Basic description of the CPS and the data used           | 1     |
|                         | Reasoning why data is valuable for PM or CM              | 2     |
| Algorithms and modeling | No description of the methods used                       | 0     |
|                         | Basic description of the methods used                    | 1     |
|                         | Reasoning why methods are used for that specific problem | 2     |
| Empirical evaluation    | No evaluation of the developed methods                   | 0     |
|                         | Basic empirical evaluation of the methods used           | 1     |
|                         | Reasoning about the performance of the methods           | 2     |

### 4.1 Descriptive analysis

Following the process of filtering the publications as depicted in Figure 1, we were left with 144 publications that fulfill our criteria (Sections 3.3, 3.4, 3.5). This section includes an analysis of how much research was done in the field of CM or PM for CPS.

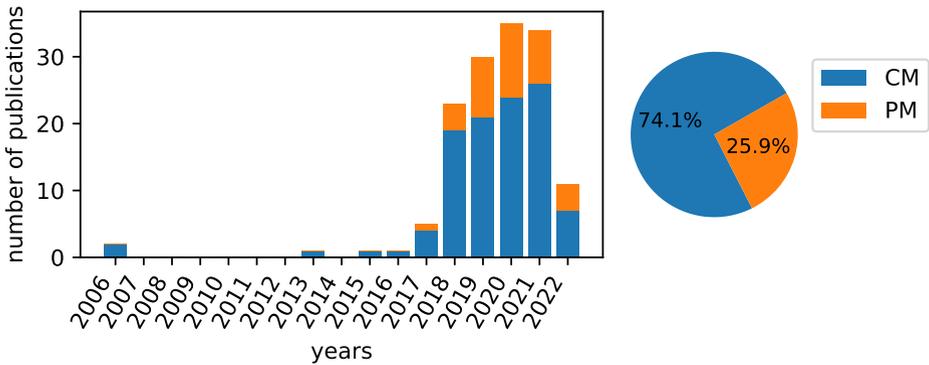


Fig. 2: Number of publications of high quality per year addressing CM or PM

The number of publications addressing CM or PM per year is shown in Figure 2. In general, papers addressing CM problems are published more often than papers addressing PM. The first CM publication was made in the year 2006, no publications matching our criteria were made from 2007 to 2012. Just three publications about CM were made in the years 2013, 2015, and 2016. In 2014, we found no publication of high quality about CM or PM for CPS. The first publications about PM were made in the year 2017. Starting in 2017, the number of publications about CM and PM increased heavily, such that the number of publications reached its maximum in the year 2020 to a total number of 35 publications. Since our study was done in the first half of 2022, the number of publications in the year 2022 is very low and not representative of a new potential trend.

## 4.2 Qualitative analysis

The final set of publications with a quality score of higher than 3 was also used for an in-depth analysis to answer the research questions (see Table 1). For that, we analyzed the full texts of each of the publications and extracted the **CPS area** (RQ1), **monitored data** (RQ1), **feature engineering technique** (RQ2), **machine learning type** (RQ3), and **CM or PM** (RQ3).

### 4.2.1 RQ1: What kind of data is used for monitoring and predicting the health of a CPS?

For evaluating the health of a specific system or to predict future system behavior it is required to gather data coming from that system by using different kinds of sensors. Most of the sensors interact with the environment and produce an electrical signal, but very often the electrical signal stands for a different physical unit. Depending on the sensor type, the electrical signal coming from the sensor is converted into the respective physical unit (i.e., temperature, acceleration, acoustics) that is monitored and used for CM or PM.

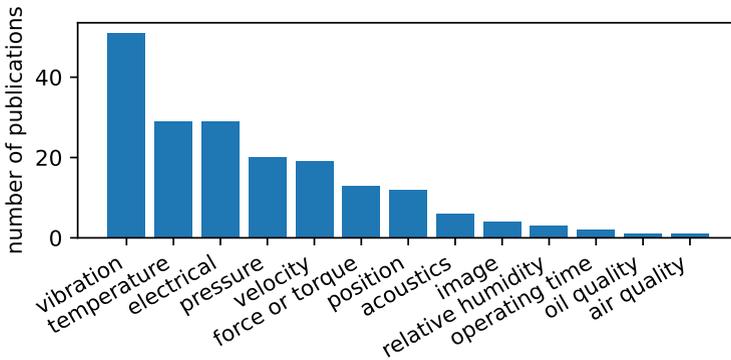


Fig. 3: Number of monitored data usages

A total of 51 publications from different domains use vibration data, e.g. [AH21, ANA20, Ki18b, Zh18]. 29 studies report on the successful use of temperature data, e.g. [CL20, Le20], also, 29 publications use electrical data, e.g. [EW18, GL18], 20 publications use pressure data, e.g. [Li18a, Ma21a], 19 publications analyze velocity data, e.g. [Bo21, LW19], 13 publications use either a force or torque as input, e.g. [Li20b, Sh21] and twelve publications use a specific position of the CPS [Ma21a, SG20]. Few papers report on the use of acoustics [Wu21], images [Vi19], relative humidity [Sy18], oil quality [Li19a], or air quality [Sy18]. Very often, a publication does not just monitor a single signal but combines different properties to a multivariate dataset, for instance, the authors of [Ma21b] combined

temperatures, velocities, torques, and pressures from an industrial press to a joint monitoring dataset.

#### 4.2.2 RQ2: What methods exist to extract meaningful features from data provided by CPS?

Data coming from CPS may contain noise that could lead to poor learning performance if not properly handled. Additionally, the high dimensionality of CPS data may lead to potential dropping performances. Due to these problems, it is very often required to not directly work on the data, but to extract meaningful features from the data and apply algorithms to the extracted features. We, therefore, identified feature extraction techniques that are successfully applied to CPS data. Figure 4 shows which feature extraction techniques are applied to what type of monitored data.

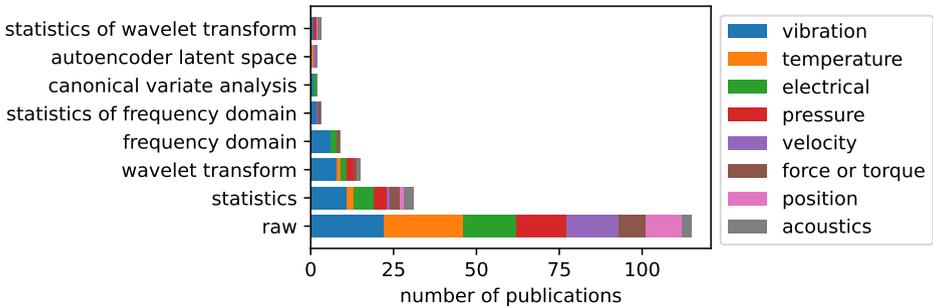


Fig. 4: Feature extracting methods with respect to the monitored data that are used more than once

Most of the publications do not use a feature extraction method, but they are applying machine learning algorithms directly to the recorded data. The most frequently used feature extraction method is to split the data into smaller segments and determine certain statistics of these segments. For example, in [DK18] the authors use the root mean square (RMS), kurtosis, crest factor, skewness, and entropy. The authors of [SZ21] calculate basic statistics (i.e., maximum, mean, root mean square, variance, standard deviation, skewness, kurtosis) from the time domain, but also from the frequency domain. These features are then combined into a common dataset as input for machine learning algorithms. The second most feature extraction method is to calculate the wavelet transform of the monitored signal [AJW20, Ca20]. Two publications [AJW21, LTT19] also compute certain statistics of wavelet transform and use these as features. Twelve publications utilize the frequency domain of the monitored signal, either by calculating the Fourier components or the power spectral density. Eight publications use the frequency components directly as data, e.g. [Xu17], and four publications compute certain statistics from the frequency domain [Zh22]. Two publications extract features by training an Autoencoder (AE) such that the latent space representation of the monitored signal is used as a feature [Fo20, Li18b]. The authors of

[ALK21, Wa21] use canonical variate analysis for extracting features. The feature extraction stage results in a dataset consisting of multiple features for every point in time. Before applying the actual evaluation or forecasting of the system status, machine learning pipelines might contain dimensionality reduction techniques to decrease the number of features. Feature reduction techniques can also be applied directly to the monitored data because different CPS sensors tend to generate correlating signals (e.g. temperature, spinning speed). In the analyzed publications, twelve publications make use of principal component analysis, e.g. [CYM15, Ch20a, Fa20, LRN20]. Linear discriminant analysis [Na21b, KH22] is used by two publications and t-distributed stochastic neighbor embedding [Se21] by one publication.

### 4.2.3 RQ3: What methods exist for CM and PM for CPS?

For identifying the most prominent methods, publications processing either simulated data sets or real industrial case studies are analyzed. As a result of this, existing machine learning methods or algorithms were identified and evaluated according to their purpose, either PM or CM. To get a precise overview of methods and algorithms among the publications, we analyzed CM methods and PM methods separately.

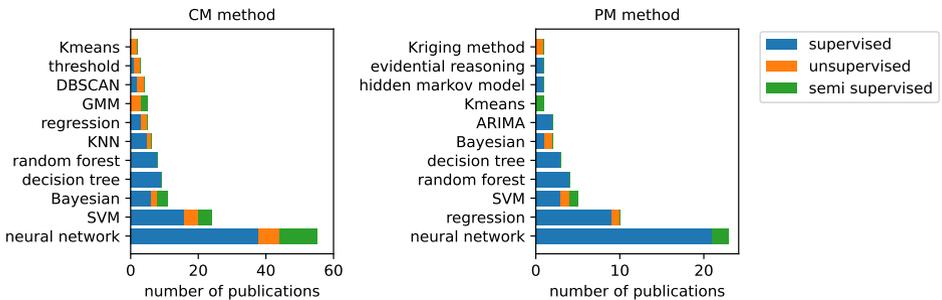


Fig. 5: Prominence of CM and PM methods

Figure 5 shows the distribution of the different methods that are used more than once for CM or PM and their general machine learning type. A general observation is that most of the described CM and PM problems are addressed by supervised learning approaches, followed by unsupervised learning and semi-supervised learning.

The machine learning technology that is used most often for both CM and PM is artificial neural networks since approximately half of the publications apply this technology in some way (e.g., deep neural networks, Convolutional Neural Networks (CNN), recurrent neural networks). Most of the publications taking advantage of neural networks use this technology for supervised learning, but neural networks are also applied in the context of unsupervised and semi-supervised learning. A more detailed overview of what kind of neural networks are utilized by the publications is given later in this section.

In the following, we concentrate on methods used in CM applications. The Support Vector

Machines (SVM) are used for supervised learning [AJW20, CCH19, GL18], unsupervised learning [BB21], and semi-supervised learning [YZ21]. Decision trees [Se18, Zh20] and random forest classifiers [Pa20, Xu19] are both mainly applied for supervised learning tasks. Different publications use algorithms that are based on Bayes' theorem (Bayesian estimation [Ly21, SG20], Bayesian filtering [FT21], Bayesian classification [EW18]). Six publications use different regression-based technologies, for instance, linear regression [Du20], polynomial regression [Vi18], or support vector regression [Sh21]. Different clustering algorithms, namely Gaussian mixture models [Ma21a], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [Sy19] and K-means clustering [Na21b] are applied mainly for unsupervised and semi-supervised learning tasks. Few publications use the result of the feature extraction as an anomaly score to measure the faultiness of the respective system. By defining a specific threshold [CYM15] on that measure, the respective data is evaluated. The remaining methods that are used just once are hidden Markov model [Ki18b], hierarchical clustering [Ka19], a method based on belief rules [Yi17], AdaBoost [LN21], affinity propagation [Ha16], recursive graph model [Ch20b], and linear discriminant analysis [KH22].

In the following, we report on PM methods. Different regression-based algorithms are used second most, namely, linear regression [FHS21, Wu18], support vector regression [Kh21, Ni21a], and RANSAC regression [JZW17]. The authors of [Le19] use weighted least squares regression and feasible generalized least squares regression. In [GK20], the authors evaluate the different regression-based methods (linear, gradient boost, random forest, extra tree, AdaBoost). SVM [Fe19, GYS21, PK20, Ye19] are utilized third most. Random forest classifiers [Be19, Yu21] are used four times, and simple decision trees [Ca20] three times, both just for supervised learning purposes. ARIMA [Ji20b] and methods that are based on Bayes' theorem [Li19b] each are used two times. K-means [Li18c], hidden Markov model [Wu18], and Kriging method [Li19b] each are utilized once.

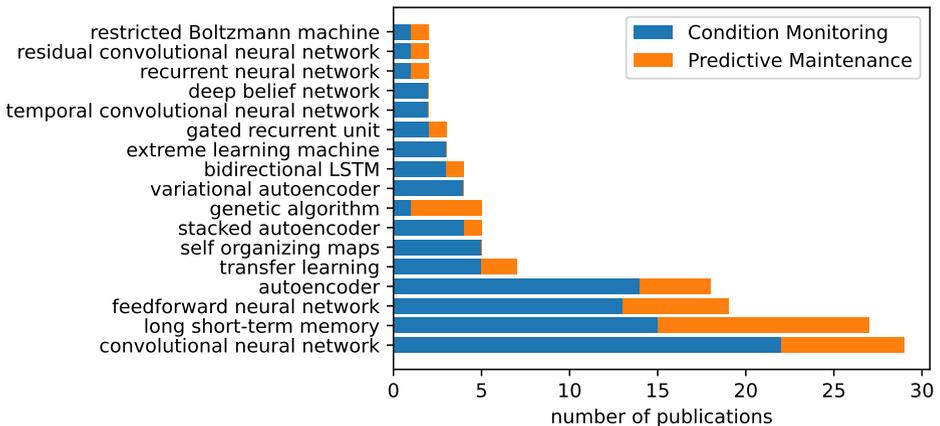


Fig. 6: Popularity of neural network types for CM and PM publications. Different architectural choices are counted individually

An overview of the neural network types used for CM and PM applications is given in Figure 6. Most of the neural networks contain either convolutional layers (*CM* [LRN20, Ni21b], *PM* [MK20, Ye19]) or LSTM cells (*CM* [TC19, VEN20], *PM* [AJW21, KC21, NZU20]). More PM publications use LSTM neural networks than convolutional-based neural networks. Pure feedforward neural networks are addressed by thirteen CM publications [Ad20, MPD18] and by six PM publications [Fa20]. Four publications use autoencoder for PM [MK20, Ye19] and 14 publications use autoencoder for CM [BB21, DK18, FG21, YZ21]. The remaining neural network technologies are more special and used less. The remaining technologies are transfer learning (*CM* [Ci21, Zh21a], *PM* [Kh21]), self-organizing maps (*CM* [Bi18, K.18, Li18a]), stacked autoencoder (*CM* [Al20, DK18], *PM* [Fo20]), genetic algorithms (*CM* [Ad20], *PM* [Fa20, KC21]), variational autoencoder (*CM* [Li18b, YZ21]), bidirectional LSTM (*CM* [So21], *PM* [Kh21]), extreme learning machine (*CM* [Xu17]), gated recurrent unit (*CM* [Zh21a], *PM* [Wi20]), temporal convolutional neural network (*CM* [S.19]), deep belief network (*CM* [Zh19]), basic recurrent neural network (*CM* [Li20a], *PM* [Ji20b]), residual convolutional neural network (*CM* [Ni21b], *PM* [MK20]), and restricted Boltzmann machines (*CM* [De22], *PM* [Fo20]).

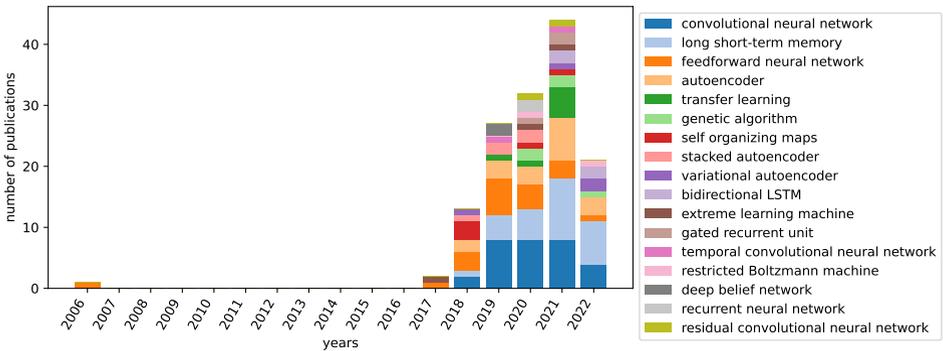


Fig. 7: Yearly distribution of techniques associated with neural networks

Figure 7 displays how the popularity of techniques that can be associated with neural networks for the purpose of PM or CM for CPS develops over the last decades. The developments give a good indication of which new trends are on the horizon and help to understand what techniques were successfully applied over a longer period of time. CNN, LSTM, and pure feedforward neural networks are applied over the longest period of time. The number of pure feedforward neural network appearances decreased after 2019. Self-organizing maps were mainly used in the year 2018 and deep belief networks were only used in 2019. The number of AE usages grows from 2018 until now. The last appearance of a stacked AE was in the year 2020 while the number of VAEs increased in recent years. The number of bidirectional long short-term memory usage grows starting in 2021. Transfer learning, extreme learning machines, and genetic algorithms are techniques that address the training process of neural networks. The number of usages increases over time.

## 5 Conclusion

The goal of this study was to report on state-of-the-art methods that are used for CM and PM tasks to fill the data engineering pipeline consisting of feature extraction and modeling. Our research questions are phrased such that we get an overview of methods that can be applied to a big variety of CPS. That was necessary since the optical synchronization system is a collection of several types of CPS. We came up with a list of publications, their addressed monitored data, feature extraction methods, and CM and PM methods.

The first research question is answered with a list of what kind of CPS data is addressed by CM and PM. Especially, CPS from different application areas producing vibration data are considered a lot. For the optical synchronization system, potential vibration sources exist such as stepper motors or water pumps. Therefore, it is planned to use accelerometers to directly identify vibration sources and apply the methods found.

The second research question addresses the topic of feature engineering. Most of the publications apply algorithms directly using the recorded signals. The identified feature extraction methods focus either on statistical analysis or on features coming from the frequency domain. The optical synchronization system can make use of that because the operators are heavily using the frequency domain of key signals for evaluating the health status of the system.

The third research question asks for CM and PM techniques. The main difference between PM and CM publications is, that CM uses more fault detection methods like clustering or anomaly detection while PM uses more regression-based algorithms. Also, the percentage of recurrent neural networks, including long short-term memory is higher among the PM publications compared to CM publications. This is because PM techniques are more likely to address the time-dependent behavior compared to CM techniques, which is a typical characteristic of recurrent neural networks.

The development of neural network-related techniques shows that recent publications tend to use more specialized learning algorithms like bidirectional LSTM or transfer learning. This shows that better planning in the neural network design reduces the costs of training huge neural networks with a simple structure.

Predictive maintenance often requires prior knowledge to build a model capable to predict future system states. Therefore, applying predictive maintenance techniques includes a manual inspection and monitoring of the system state over a longer time.

In conclusion, the review of existing PM and CM work builds an extremely helpful foundation for systematically evaluating the health status of the optical synchronization system and predicting future system outages.

## Bibliography

- [AA21] Ali, Zainab H.; Ali, Hesham A.: Towards sustainable smart IoT applications architectural elements and design: opportunities, challenges, and open directions. *The Journal of Supercomputing*, 77(6):5668–5725, 2021.
- [AC17] Aminikhanghahi, Samaneh; Cook, Diane J: A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.
- [AC20] Ali, Syed Saqib; Choi, Bong Jun: State-of-the-Art Artificial Intelligence Techniques for Distributed Smart Grids: A Review. *Electronics*, 9(6):1030, 2020.
- [Ad20] Adnan, Ahmed; Muhammed, Abdullah; Abd Ghani, Abdul Azim; Abdullah, Azizol; Hakim, Fahrul: Hyper-Heuristic Framework for Sequential Semi-Supervised Classification Based on Core Clustering. *Symmetry*, 12(8):1292, 2020.
- [AH20] Adedeji, Kazeem B.; Hamam, Yskandar: Cyber-Physical Systems for Water Supply Network Management: Basics, Challenges, and Roadmap. *Sustainability*, 12(22):9555, 2020.
- [AH21] Akpudo, Ugochukwu Ejike; Hur, Jang-Wook: D-dCNN: A Novel Hybrid Deep Learning-Based Tool for Vibration-Based Diagnostics. *Energies*, 14(17):5286, 2021.
- [AJW20] Akpudo, Ugochukwu Ejike; Jang-Wook, Hur: A Multi-Domain Diagnostics Approach for Solenoid Pumps Based on Discriminative Features. *IEEE Access*, 8:175020–175034, 2020.
- [AJW21] Akpudo, Ugochukwu Ejike; Jang-Wook, Hur: An Automated Sensor Fusion Approach for the RUL Prediction of Electromagnetic Pumps. *IEEE Access*, 9:38920–38933, 2021.
- [AKI21] Abid, Anam; Khan, Muhammad Tahir; Iqbal, Javaid: A review on fault detection and diagnosis techniques: basics and beyond. *Artificial Intelligence Review*, 54(5):3639–3664, 2021.
- [AI20] Alo, Uzoma Rita; Nweke, Henry Friday; Teh, Ying Wah; Murtaza, Ghulam: Smartphone Motion Sensor-Based Complex Human Activity Identification Using Deep Stacked Autoencoder Algorithm for Enhanced Smart Healthcare System. *Sensors*, 20(21), 2020.
- [ALK21] Agron, Danielle Jaye S.; Lee, Jae-Min; Kim, Dong-Seong: Nozzle Thermal Estimation for Fused Filament Fabricating 3D Printer Using Temporal Convolutional Neural Networks. *Applied Sciences*, 11(14):6424, 2021.
- [ANA20] Anagiannis, Ioannis; Nikolakis, Nikolaos; Alexopoulos, Kosmas: Energy-Based Prognosis of the Remaining Useful Life of the Coating Segments in Hot Rolling Mill. *Applied Sciences*, 10(19):6827, 2020.
- [Ar21] Arooj, Ansif; Farooq, Muhammad Shoaib; Akram, Aftab; Iqbal, Razi; Sharma, Ashutosh; Dhiman, Gaurav: Big Data Processing and Analysis in Internet of Vehicles: Architecture, Taxonomy, and Open Research Challenges. *Archives of Computational Methods in Engineering*, pp. 1–37, 2021.
- [BB21] Bulla, Chetan; Birje, Mahantesh N.: Improved data-driven root cause analysis in fog computing environment. *Journal of Reliable Intelligent Environments*, 2021.

- [BCC21] Bansal, Maggi; Chana, Inderveer; Clarke, Siobhán: A Survey on IoT Big Data. *ACM Computing Surveys*, 53(6):1–59, 2021.
- [Be19] Behera, Sourajit; Choubey, Anurag; Kanani, Chandresh S.; Patel, Yashwant Singh; Misra, Rajiv; Sillitti, Alberto: Ensemble trees learning based improved predictive maintenance using IIoT for turbofan engines. In (Hung, Chih-Cheng; Papadopoulos, George A., eds): *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, New York, NY, USA, pp. 842–850, 2019.
- [Bi18] von Birgelen, Alexander; Buratti, Davide; Mager, Jens; Niggemann, Oliver: Self-Organizing Maps for Anomaly Localization and Predictive Maintenance in Cyber-Physical Production Systems. *Procedia CIRP*, 72:480–485, 2018.
- [Bo19] Bousdekis, Alexandros; Lepenioti, Katerina; Apostolou, Dimitris; Mentzas, Gregoris: Decision Making in Predictive Maintenance: Literature Review and Research Agenda for Industry 4.0. *IFAC-PapersOnLine*, 52(13):607–612, 2019. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.
- [Bo21] Bolbot, Victor; Theotokatos, Gerasimos; Hamann, Rainer; Psarros, George; Boulougouris, Evangelos: Dynamic Blackout Probability Monitoring System for Cruise Ship Power Plants. *Energies*, 14(20):6598, 2021.
- [BRK21] Black, Innes Murdo; Richmond, Mark; Kolios, Athanasios: Condition monitoring systems: a systematic literature review on machine-learning methods improving offshore-wind turbine operational management. *International Journal of Sustainable Energy*, 40(10):923–946, 2021.
- [Ca20] de Carvalho Chrysostomo, Giovanni Gravito; de Aguiar Vallim, Marco Vinicius Bhering; Da Silva, Leilton Santos; Silva, Leandro A.; de Aguiar Vallim Filho, Arnaldo Rabello: A Framework for Big Data Analytical Process and Mapping—BAProM: Description of an Application in an Industrial Environment. *Energies*, 13(22):6014, 2020.
- [CCH19] Chang, Ching-Yuan; Chang, En-Chieh; Huang, Chi-Wen: In Situ Diagnosis of Industrial Motors by Using Vision-Based Smart Sensing Technology. *Sensors (Basel, Switzerland)*, 19(24):5340, 2019.
- [Ch20a] Choudhary, Gaurav; Astillo, Philip Virgil; You, Ilsun; Yim, Kangbin; Chen, Ing-Ray; Cho, Jin-Hee: Lightweight Misbehavior Detection Management of Embedded IoT Devices in Medical Cyber Physical Systems. *IEEE Transactions on Network and Service Management*, 17(4):2496–2510, 2020.
- [Ch20b] Chunlin, Guo; Chenliang, Zhang; Tao, Li; Kejia, Zhu; Huiyuan, Ma: Transformer Vibration Feature Extraction Method Based on Recursive Graph Quantitative Analysis. In: *2020 IEEE/IAS Industrial and Commercial Power System Asia (I&CPS Asia)*. pp. 1046–1049, 2020.
- [Ci21] Cipriani, Giovanni; Manno, Donatella; Dio, Vincenzo Di; Sciortino, Giovanni: Automatic detection of thermal anomalies in induction motors. In: *2021 IEEE International Conference on Environment and Electrical Engineering and 2021 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. pp. 1–6, 2021.
- [CL20] Choi, Jeonghun; Lee, Seung Jun: Consistency Index-Based Sensor Fault Detection System for Nuclear Power Plant Emergency Situations Using an LSTM Network. *Sensors (Basel, Switzerland)*, 20(6):1651, 2020.

- [CYM15] Chen, Po-Yu; Yang, Shusen; McCann, Julie A.: Distributed Real-Time Anomaly Detection in Networked Industrial Sensing Systems. *IEEE Transactions on Industrial Electronics*, 62(6):3832–3842, 2015.
- [De21] De Kooning, Jeroen D. M.; Stockman, Kurt; De Maeyer, Jeroen; Jarquin-Laguna, Antonio; Vandavelde, Lieven: Digital Twins for Wind Energy Conversion Systems: A Literature Review of Potential Modelling Techniques Focused on Model Fidelity and Computational Load. *Processes*, 9(12), 2021.
- [De22] Demertzis, Konstantinos; Iliadis, Lazaros; Pimenidis, Elias; Kikiras, Panagiotis: Variational restricted Boltzmann machines to automated anomaly detection. *Neural Computing and Applications*, pp. 1–14, 2022.
- [DK18] Duong, Bach Phi; Kim, Jong-Myon: Non-Mutually Exclusive Deep Neural Network Classifier for Combined Modes of Bearing Fault Diagnosis. *Sensors (Basel, Switzerland)*, 18(4):1129, 2018.
- [Dr21] Drakaki, Maria; Karnavas, Yannis L.; Tzionas, Panagiotis; Chasiotis, Ioannis D.: Recent Developments Towards Industry 4.0 Oriented Predictive Maintenance in Induction Motors. *Procedia Computer Science*, 180:943–949, 2021.
- [dSGC22] de Santis, Rodrigo Barbosa; Gontijo, Tiago Silveira; Costa, Marcelo Azevedo: Condition-based maintenance in hydroelectric plants: A systematic literature review. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 236(5):631–646, 2022.
- [Du20] Dutta, Nabanita; Palanisamy, Kaliannan; Subramaniam, Umashankar; Padmanaban, Sanjeevikumar; Holm-Nielsen, Jens Bo; Blaabjerg, Frede; Almakhles, Dhafer Jaber: Identification of Water Hammering for Centrifugal Pump Drive Systems. *Applied Sciences*, 10(8):2683, 2020.
- [EW18] E. F. Swana; W. Doorsamy: Fault Diagnosis on a Wound Rotor Induction Generator Using Probabilistic Intelligence. In: 2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe). pp. 1–5, 2018.
- [Fa20] Farooq, Basit; Bao, Jinsong; Li, Jie; Liu, Tianyuan; Yin, Shiyong: Data-Driven Predictive Maintenance Approach for Spinning Cyber-Physical Production System. *Journal of Shanghai Jiaotong University (Science)*, 25(4):453–462, 2020.
- [Fe19] Ferrero Bermejo, Jesús; Gómez Fernández, Juan Francisco; Pino, Rafael; Crespo Márquez, Adolfo; Guillén López, Antonio Jesús: Review and Comparison of Intelligent Optimization Modelling Techniques for Energy Forecasting and Condition-Based Maintenance in PV Plants. *Energies*, 12(21):4163, 2019.
- [FG21] Favarelli, Elia; Giorgetti, Andrea: Machine Learning for Automatic Processing of Modal Analysis in Damage Detection of Bridges. *IEEE Transactions on Instrumentation and Measurement*, 70:1–13, 2021.
- [FHS21] Ferdousi, Rahatara; Hossain, M. Anwar; Saddik, Abdulmotaleb El: Early-Stage Risk Prediction of Non-Communicable Disease Using Machine Learning in Health CPS. *IEEE Access*, 9:96823–96837, 2021.

- [Fo20] Fotiadou, Konstantina; Velivassaki, Terpsichori Helen; Voulkidis, Artemis; Skias, Dimitrios; de Santis, Corrado; Zahariadis, Theodore: Proactive Critical Energy Infrastructure Protection via Deep Feature Learning. *Energies*, 13(10):2622, 2020.
- [FT21] Feng, Cheng; Tian, Pengwei: Time Series Anomaly Detection for Cyber-physical Systems via Neural System Identification and Bayesian Filtering. In (Zhu, Feida; Chin Ooi, Beng; Miao, Chunyan, eds): *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, USA, pp. 2858–2867, 2021.
- [GK20] Gupta, Akshita; Kumar, Arun: Mid Term Daily Load Forecasting using ARIMA, Wavelet-ARIMA and Machine Learning. In: *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. pp. 1–5, 2020.
- [GL18] Guo, Han; Liu, Meng-Kun: Induction motor faults diagnosis using support vector machine to the motor current signature. In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, pp. 417–421, 2018.
- [GYS21] Görür, Orhan Can; Yu, Xin; Sivrikaya, Fikret: Integrating Predictive Maintenance in Adaptive Process Scheduling for a Safe and Efficient Industrial Process. *Applied Sciences*, 11(11):5042, 2021.
- [Ha11] Hashemian, H. M.: State-of-the-Art Predictive Maintenance Techniques. *IEEE Transactions on Instrumentation and Measurement*, 60(1):226–236, 2011.
- [Ha16] Haeckell, Moritz W.; Rolfes, Raimund; Kane, Michael B.; Lynch, Jerome P.: Three-Tier Modular Structural Health Monitoring Framework Using Environmental and Operational Condition Clustering for Data Normalization: Validation on an Operational Wind Turbine System. *Proceedings of the IEEE*, 104(8):1632–1646, 2016.
- [HB21] Hassankhani Dolatabadi, Sepideh; Budinska, Ivana: Systematic Literature Review Predictive Maintenance Solutions for SMEs from the Last Decade. *Machines*, 9(9):191, 2021.
- [Hu21] Huang, Ziqi; Shen, Yang; Li, Jiayi; Fey, Marcel; Brecher, Christian: A Survey on AI-Driven Digital Twins in Industry 4.0: Smart Manufacturing and Advanced Robotics. *Sensors (Basel, Switzerland)*, 21(19):6340, 2021.
- [ITK19] Ismail, Ahmed; Truong, Hong-Linh; Kastner, Wolfgang: Manufacturing process data analysis pipelines: a requirements analysis and survey. *Journal of Big Data*, 6(1), 2019.
- [Ji20a] Ji, Shunhui; Li, Qingqiu; Cao, Wennan; Zhang, Pengcheng; Muccini, Henry: Quality Assurance Technologies of Big Data Applications: A Systematic Literature Review. *Applied Sciences*, 10(22):8052, 2020.
- [Ji20b] Jimenez-Cortadi, Alberto; Irigoien, Itziar; Boto, Fernando; Sierra, Basilio; Rodriguez, German: Predictive Maintenance on the Machining Process and Machine Tool. *Applied Sciences*, 10(1), 2020.
- [JZW17] Jung, Deokwoo; Zhang, Zhenjie; Winslett, Marianne: Vibration Analysis for IoT Enabled Predictive Maintenance. In: *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. pp. 1271–1282, 2017.

- [K.18] K. Amarasinghe; C. Wickramasinghe; D. Marino; C. Rieger; M. Manic: Framework for Data Driven Health Monitoring of Cyber-Physical Systems. In: 2018 Resilience Week (RWS). pp. 25–30, 2018.
- [Ka19] Kawatsu, Kaname: PHM by Using Multi-Physics System-Level Modeling and Simulation for EMAs of Liquid Rocket Engine. In: 2019 IEEE Aerospace Conference. IEEE, pp. 1–10, 2019.
- [KC07] Kitchenham, Barbara; Charters, Stuart: Guidelines for performing Systematic Literature Reviews in Software Engineering. 2007.
- [KC21] Kim, Do-Gyun; Choi, Jin-Young: Optimization of Design Parameters in LSTM Model for Predictive Maintenance. *Applied Sciences*, 11(14):6450, 2021.
- [Kh21] Khan, Noman; Ullah, Fath U Min; Afnan; Ullah, Amin; Lee, Mi Young; Baik, Sung Wook: Batteries State of Health Estimation via Efficient Neural Networks With Multiple Channel Charging Profiles. *IEEE Access*, 9:7797–7813, 2021.
- [KH22] Kim, Doyun; Heo, Tae-Young: Anomaly Detection with Feature Extraction Based on Machine Learning Using Hydraulic System IoT Sensor Data. *Sensors*, 22(7), 2022.
- [Ki18a] Kim, Dong-Hyeon; Kim, Thomas J. Y.; Wang, Xinlin; Kim, Mincheol; Quan, Ying-Jun; Oh, Jin Woo; Min, Soo-Hong; Kim, Hyungjung; Bhandari, Binayak; Yang, Insoon; Ahn, Sung-Hoon: Smart Machining Process Using Machine Learning: A Review and Perspective on Machining Industry. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 5(4):555–568, 2018.
- [Ki18b] Kißkalt, Dominik; Fleischmann, Hans; Kreitlein, Sven; Knott, Manuel; Franke, Jörg: A novel approach for data-driven process and condition monitoring systems on the example of mill-turn centers. *Production Engineering*, 12(3-4):525–533, 2018.
- [KM21] Kostrzewski, Mariusz; Melnik, Rafał: Condition Monitoring of Rail Transport Systems: A Bibliometric Performance Analysis and Systematic Literature Review. *Sensors*, 21(14), 2021.
- [Le18] Lee, Gil-Yong; Kim, Mincheol; Quan, Ying-Jun; Kim, Min-Sik; Kim, Thomas Joon Young; Yoon, Hae-Sung; Min, Sangkee; Kim, Dong-Hyeon; Mun, Jeong-Wook; Oh, Jin Woo; Choi, In Gyu; Kim, Chung-Soo; Chu, Won-Shik; Yang, Jinkyu; Bhandari, Binayak; Lee, Choon-Man; Ihn, Jeong-Beom; Ahn, Sung-Hoon: Machine health management in smart factory: A review. *Journal of Mechanical Science and Technology*, 32(3):987–1009, 2018.
- [Le19] Lee, Chia-Yen; Huang, Ting-Syun; Liu, Meng-Kun; Lan, Chen-Yang: Data Science for Vibration Heteroscedasticity and Predictive Maintenance of Rotary Bearings. *Energies*, 12(5):801, 2019.
- [Le20] Letzgus, Simon: Change-point detection in wind turbine SCADA data for robust condition monitoring with normal behaviour models. *Wind Energy Science*, 5(4):1375–1397, 2020.
- [Li18a] Li, Juanli; Xie, Jiacheng; Yang, Zhaojian; Li, Junjie: Fault Diagnosis Method for a Mine Hoist in the Internet of Things Environment. *Sensors (Basel, Switzerland)*, 18(6):1920, 2018.
- [Li18b] Liao, Weixian; Guo, Yifan; Chen, Xuhui; Li, Pan: A Unified Unsupervised Gaussian Mixture Variational Autoencoder for High Dimensional Outlier Detection. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, pp. 1208–1217, 2018.

- [Li18c] Liu, Zongchang; Jin, Chao; Jin, Wenjing; Lee, Jay; Zhang, Zhiqiang; Peng, Chang; Xu, Guanji: Industrial AI Enabled Prognostics for High-speed Railway Systems. In: 2018 IEEE International Conference on Prognostics and Health Management (ICPHM). pp. 1–8, 2018.
- [Li19a] Li, Bao-rui; Wang, Yi; Dai, Guo-hong; Wang, Ke-sheng: Framework and case study of cognitive maintenance in Industry 4.0. *Frontiers of Information Technology & Electronic Engineering*, 20(11):1493–1504, 2019.
- [Li19b] Liu, Hanbing; He, Xin; Jiao, Yubo; Wang, Xirui: Reliability Assessment of Deflection Limit State of a Simply Supported Bridge using vibration data and Dynamic Bayesian Network Inference. *Sensors*, 19(4), 2019.
- [Li20a] Ling, Jun; Liu, Gao-Jun; Li, Jia-Liang; Shen, Xiao-Cheng; You, Dong-Dong: Fault prediction method for nuclear power machinery based on Bayesian PPCA recurrent neural network model. *Nuclear Science and Techniques*, 31(8), 2020.
- [Li20b] Liu, Meng-Kun; Tran, Minh-Quang; Chung, Chunhui; Qui, Yi-Wen: Hybrid model- and signal-based chatter detection in the milling process. *Journal of Mechanical Science and Technology*, 34(1):1–10, 2020.
- [Li21] Lima, André Luis da Cunha Dantas; Aranha, Vitor Moraes; Carvalho, Caio Jordão de Lima; Nascimento, Erick Giovanni Sperandio: Smart predictive maintenance for high-performance computing systems: a literature review. *The Journal of Supercomputing*, 77(11):13494–13513, 2021.
- [LN21] Li, Peng; Niggemann, Oliver: A Nonconvex Archetypal Analysis for One-Class Classification Based Anomaly Detection in Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics*, 17(9):6429–6437, 2021.
- [LRN20] Langarica, Saul; Ruffelmacher, Christian; Nunez, Felipe: An Industrial Internet Application for Real-Time Fault Diagnosis in Industrial Motors. *IEEE Transactions on Automation Science and Engineering*, 17(1):284–295, 2020.
- [LTT19] Liu, Meng-Kun; Tseng, Yi-Heng; Tran, Minh-Quang: Tool wear monitoring and prediction based on sound signal. *The International Journal of Advanced Manufacturing Technology*, 103(9):3361–3373, 2019.
- [LW19] Li, Meng; Wang, Shuangxin: Dynamic Fault Monitoring of Pitch System in Wind Turbines using Selective Ensemble Small-World Neural Networks. *Energies*, 12(17):3256, 2019.
- [Ly21] Lyu, Dongzhen; Niu, Guangxing; Yang, Tao; Gang, Chen; Zhang, Bin: Uncertainty Analysis in the Application of Fault Diagnosis and Prognosis. In: 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS). IEEE, pp. 686–690, 2021.
- [Ma20] Maldonado-Correa, Jorge; Martín-Martínez, Sergio; Artigao, Estefanía; Gómez-Lázaro, Emilio: Using SCADA data for wind turbine condition monitoring: A systematic literature review. *Energies*, 13(12):3132, 2020.
- [Ma21a] Maseda, F. Javier; López, Iker; Martija, Itziar; Alkorta, Patxi; Garrido, Aitor J.; Garrido, Izaskun: Sensors Data Analysis in Supervisory Control and Data Acquisition (SCADA) Systems to Foresee Failures with an Undetermined Origin. *Sensors (Basel, Switzerland)*, 21(8):2762, 2021.

- [Ma21b] Mateus, Balduino César; Mendes, Mateus; Farinha, José Torres; Cardoso, António Marques: Anticipating Future Behavior of an Industrial Press Using LSTM Networks. *Applied Sciences*, 11(13):6101, 2021.
- [MK20] Meesublak, Koonlachat; Klinsukont, Tosapol: A Cyber-Physical System Approach for Predictive Maintenance. In: 2020 IEEE International Conference on Smart Internet of Things (SmartIoT). IEEE, pp. 337–341, 2020.
- [MPD18] Majdani, Farzan; Petrovski, Andrei; Doolan, Daniel: Evolving ANN-based sensors for a context-aware cyber physical system of an offshore gas turbine. *Evolving Systems*, 9(2):119–133, 2018.
- [MTT21] Maktoubian, Jamal; Taskhiri, Mohammad Sadegh; Turner, Paul: Intelligent Predictive Maintenance (IPdM) in Forestry: A Review of Challenges and Opportunities. *Forests*, 12(11):1495, 2021.
- [Na21a] Nacchia, Milena; Fruggiero, Fabio; Lambiase, Alfredo; Bruton, Ken: A Systematic Mapping of the Advancing Use of Machine Learning Techniques for Predictive Maintenance in the Manufacturing Sector. *Applied Sciences*, 11(6):2546, 2021.
- [Na21b] Naik, Kshirasagar; Pandey, Mahesh D.; Panda, Anannya; Albasir, Abdurhman; Taneja, Kunal: Data Driven Modelling of Nuclear Power Plant Performance Data as Finite State Machines. *Modelling*, 2(1):43–62, 2021.
- [Ni21a] Nie, Shouren; Jiang, Yuchen; Li, Kuan; Luo, Hao; Li, Xianling; Wu, Yunkai: Remaining useful life prediction approach for rolling element bearings based on optimized SVR model with reliable time intervals. In: 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS). IEEE, pp. 673–678, 2021.
- [Ni21b] Niu, Guangxing; Liu, Enhui; Zhang, Bin; Golda, Michael; Mastro, Stephen: A Deep Residual Convolutional Neural Network based Bearing Fault Diagnosis with Multi-Sensor Data. In: 2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS). IEEE, pp. 655–660, 2021.
- [NUS21a] Nath, Aneesh G.; Udmale, Sandeep S.; Singh, Sanjay Kumar: Role of artificial intelligence in rotor fault diagnosis: a comprehensive review. *Artificial Intelligence Review*, 54(4):2609–2668, 2021.
- [NUS21b] Nath, Aneesh G.; Udmale, Sandeep S.; Singh, Sanjay Kumar: Role of artificial intelligence in rotor fault diagnosis: A comprehensive review. *Artificial Intelligence Review*, 54(4):2609–2668, 2021.
- [NZU20] Niyonambaza, Irene; Zennaro, Marco; Uwitonze, Alfred: Predictive Maintenance (PdM) Structure Using Internet of Things (IoT) for Mechanical Equipment Used into Hospitals in Rwanda. *Future Internet*, 12(12):224, 2020.
- [Pa20] Panicucci, Simone; Nikolakis, Nikolaos; Cerquitelli, Tania; Ventura, Francesco; Proto, Stefano; Macii, Enrico; Makris, Sotiris; Bowden, David; Becker, Paul; O'Mahony, Niamh; Morabito, Lucrezia; Napione, Chiara; Marguglio, Angelo; Coppo, Guido; Andolina, Salvatore: A Cloud-to-Edge Approach to Support Predictive Analytics in Robotics Industry. *Electronics*, 9(3):492, 2020.
- [PK20] Pandit, Ravi; Kolios, Athanasios: SCADA Data-Based Support Vector Machine Wind Turbine Power Curve Uncertainty Estimation and Its Comparative Studies. *Applied Sciences*, 10(23), 2020.

- [PVB21] Pech, Martin; Vrchota, Jaroslav; Bednář, Jiří: Predictive Maintenance and Intelligent Sensors in Smart Factory: Review. *Sensors* (Basel, Switzerland), 21(4), 2021.
- [RTJ21a] Rinaldi, Giovanni; Thies, Philipp R.; Johanning, Lars: Current Status and Future Trends in the Operation and Maintenance of Offshore Wind Turbines: A Review. *Energies*, 14(9):2484, 2021.
- [RTJ21b] Rinaldi, Giovanni; Thies, Philipp R.; Johanning, Lars: Current Status and Future Trends in the Operation and Maintenance of Offshore Wind Turbines: A Review. *Energies*, 14(9), 2021.
- [S.19] S. Afrasiabi; M. Afrasiabi; B. Parang; M. Mohammadi; M. M. Arefi; M. Rastegar: Wind Turbine Fault Diagnosis with Generative-Temporal Convolutional Neural Network. In: 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe). pp. 1–5, 2019.
- [Sc19] Schulz, Sebastian; Czwalinna, Marie; Felber, Matthias; Fenner, Michael; Gerth, Christopher; Kozak, Tomasz; Lamb, Thorsten; Lautenschlager, Björn; Ludwig, Frank; Mavrič, Uros; Müller, Jost; Pfeiffer, Sven; Schlarb, Holger; Schmidt, Christian; Sydlo, Cezary; Tiberidze, Mikheil; Zummack, Falco, eds. *Few-Femtosecond Facility-Wide Synchronization of the European XFEL*: JACoW Publishing, Geneva, Switzerland, 2019.
- [Se18] Sezer, Erim; Romero, David; Guedea, Federico; Macchi, Marco; Emmanouilidis, Christos: An Industry 4.0-Enabled Low Cost Predictive Maintenance Approach for SMEs. In: 2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC). pp. 1–8, 2018.
- [Se21] Serradilla, Oscar; Zugasti, Ekhi; Ramirez de Okariz, Julian; Rodriguez, Jon; Zurutuza, Urko: Adaptable and Explainable Predictive Maintenance: Semi-Supervised Deep Learning for Anomaly Detection and Diagnosis in Press Machine Data. *Applied Sciences*, 11(16), 2021.
- [Se22] Serradilla, Oscar; Zugasti, Ekhi; Rodriguez, Jon; Zurutuza, Urko: Deep learning models for predictive maintenance: a survey, comparison, challenges and prospects. *Applied Intelligence*, pp. 1–31, 2022.
- [SG20] Shangguan, Lantian; Gopalswamy, Swaminathan: Health Monitoring for Cyber Physical Systems. *IEEE Systems Journal*, 14(1):1457–1467, 2020.
- [Sh21] Sheuly, Sharmin Sultana; Barua, Shaibal; Begum, Shahina; Ahmed, Mobyen Uddin; Güclü, Ekrem; Osbakk, Michael: Data analytics using statistical methods and machine learning: a case study of power transfer units. *The International Journal of Advanced Manufacturing Technology*, 114(5-6):1859–1870, 2021.
- [So20] Sobolev, Egor; Zolotarev, Sergei; Giewekemeyer, Klaus; Bielecki, Johan; Okamoto, Kenta; Reddy, Hemanth K. N.; Andreasson, Jakob; Ayyer, Kartik; Barak, Imrich; Bari, Sadia; Barty, Anton; Bean, Richard; Bobkov, Sergey; Chapman, Henry N.; Chojnowski, Grzegorz; Daurer, Benedikt J.; Dörner, Katerina; Ekeberg, Tomas; Flückiger, Leonie; Galzitskaya, Oxana; Gelisio, Luca; Hauf, Steffen; Hogue, Brenda G.; Horke, Daniel A.; Hosseinizadeh, Ahmad; Ilyin, Vyacheslav; Jung, Chulho; Kim, Chan; Kim, Yoonhee; Kirian, Richard A.; Kirkwood, Henry; Kulyk, Olena; Küpper, Jochen; Letrun, Romain; Loh, N. Duane; Lorenzen, Kristina; Messerschmidt, Marc; Mühlig, Kerstin; Ourmazd, Abbas; Raab, Natascha; Rode, Andrei V.; Rose, Max; Round, Adam; Sato, Takushi; Schubert,

- Robin; Schwander, Peter; Sellberg, Jonas A.; Sikorski, Marcin; Silenzi, Alessandro; Song, Changyong; Spence, John C. H.; Stern, Stephan; Sztuk-Dambietz, Jolanta; Teslyuk, Anthon; Timneanu, Nicusor; Trebbin, Martin; Uetrecht, Charlotte; Weinhausen, Britta; Williams, Garth J.; Xavier, P. Lourdu; Xu, Chen; Vartanyants, Ivan A.; Lamzin, Victor S.; Mancuso, Adrian; Maia, Filipe R. N. C.: Megahertz single-particle imaging at the European XFEL. *Communications Physics*, 3(1), 2020.
- [So21] Song, Lin; Wang, Liping; Wu, Jun; Liang, Jianhong; Liu, Zhigui: Integrating Physics and Data Driven Cyber-Physical System for Condition Monitoring of Critical Transmission Components in Smart Production Line. *Applied Sciences*, 11(19), 2021.
- [Sy18] Syafrudin, Muhammad; Alfian, Ganjar; Fitriyani, Norma Latif; Rhee, Jongtae: Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time Monitoring System in Automotive Manufacturing. *Sensors (Basel, Switzerland)*, 18(9):2946, 2018.
- [Sy19] Syafrudin, Muhammad; Fitriyani, Norma; Alfian, Ganjar; Rhee, Jongtae: An Affordable Fast Early Warning System for Edge Computing in Assembly Line. *Applied Sciences*, 9(1):84, 2019.
- [SYD11] Sharma, Anil; Yadava, GS; Deshmukh, SG: A literature review and future perspectives on maintenance optimization. *Journal of Quality in Maintenance Engineering*, 2011.
- [SZ21] Sundaram, Sarvesh; Zeid, Abe: Smart Prognostics and Health Management (SPHM) in Smart Manufacturing: An Interoperable Framework. *Sensors (Basel, Switzerland)*, 21(18):5994, 2021.
- [TC19] Tsai, Chien-De; Chiu, Ming-Chuan: Apply Machine Learning to Improve Fault Detection and Classification in Cyber Physical System. In (Hiekata, Kazuo; Moser, Bryan R.; Inoue, Masato; Stjepandić, Josip; Wognum, Nel, eds): *Transdisciplinary Engineering for Complex Socio-technical Systems, Advances in Transdisciplinary Engineering*. IOS Press, 2019.
- [VEN20] Vos, Carlo; Eiteneuer, Benedikt; Niggemann, Oliver: Incorporating Uncertainty into Unsupervised Machine Learning for Cyber-Physical Systems. In: *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*. IEEE, pp. 475–480, 6/10/2020 - 6/12/2020.
- [Vi18] Villalonga, Alberto; Beruvides, Gerardo; Castaño, Fernando; Haber, Rodolfo: Industrial cyber-physical system for condition-based monitoring in manufacturing processes. In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. pp. 637–642, 2018.
- [Vi19] Villalba-Diez, Javier; Schmidt, Daniel; Gevers, Roman; Ordieres-Meré, Joaquín; Buchwitz, Martin; Wellbrock, Wanja: Deep Learning for Industrial Computer Vision Quality Control in the Printing Industry 4.0. *Sensors (Basel, Switzerland)*, 19(18):3987, 2019.
- [Wa21] Wang, Shubin; Tian, Yukun; Deng, Xiaogang; Cao, Qianlei; Wang, Lei; Sun, Pengxiang: Disturbance Detection of a Power Transmission System Based on the Enhanced Canonical Variate Analysis Method. *Machines*, 9(11):272, 2021.
- [Wi20] Wiese, Benedikt; Pedersen, Niels L.; Nadimi, Esmaeil S.; Herp, Jürgen: Estimating the Remaining Power Generation of Wind Turbines—An Exploratory Study for Main Bearing Failures. *Energies*, 13(13), 2020.

- [Wu18] Wu, Zhenyu; Luo, Hao; Yang, Yunong; Lv, Peng; Zhu, Xinning; Ji, Yang; Wu, Bian: K-PdM: KPI-Oriented Machinery Deterioration Estimation Framework for Predictive Maintenance Using Cluster-Based Hidden Markov Model. *IEEE Access*, 6:41676–41687, 2018.
- [Wu21] Wu, Huanzhuo; He, Jia; Tomoskozi, Mate; Fitzek, Frank H.P.: Abstraction-based Multi-object Acoustic Anomaly Detection for Low-complexity Big Data Analysis. In: 2021 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, pp. 1–6, 2021.
- [Xu17] Xu, Zhao; Hu, Changhua; Yang, Feng; Kuo, Shyh-Hao; Goh, Chi-Keong; Gupta, Amit; Nadarajan, Sivakumar: Data-Driven Inter-Turn Short Circuit Fault Detection in Induction Machines. *IEEE Access*, 5:25055–25068, 2017.
- [Xu19] Xu, Gaowei; Liu, Min; Jiang, Zhuofu; Söffker, Dirk; Shen, Weiming: Bearing Fault Diagnosis Method Based on Deep Convolutional Neural Network and Random Forest Ensemble Learning. *Sensors (Basel, Switzerland)*, 19(5):1088, 2019.
- [Ye19] Yeh, Chia-Hung; Lin, Min-Hui; Lin, Chien-Hung; Yu, Cheng-En; Chen, Mei-Juan: Machine Learning for Long Cycle Maintenance Prediction of Wind Turbine. *Sensors (Basel, Switzerland)*, 19(7):1671, 2019.
- [Yi17] Yin, Xiaojing; Wang, Zhanli; Zhang, Bangcheng; Zhou, Zhijie; Feng, Zhichao; Hu, Guanyu; Wei, Hang: A Double Layer BRB Model for Health Prognostics in Complex Electromechanical System. *IEEE Access*, 5:23833–23847, 2017.
- [Yu21] Yu, Hui; Chen, Chuang; Lu, Ningyun; Wang, Cunsong: Deep Auto-Encoder and Deep Forest-Assisted Failure Prognosis for Dynamic Predictive Maintenance Scheduling. *Sensors*, 21(24), 2021.
- [YZ21] Yang, Luoxiao; Zhang, Zijun: Wind Turbine Gearbox Failure Detection Based on SCADA Data: A Deep Learning-Based Approach. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11, 2021.
- [Zh18] Zhou, Funa; Hu, Po; Yang, Shuai; Wen, Chenglin: A Multimodal Feature Fusion-Based Deep Learning Method for Online Fault Diagnosis of Rotating Machinery. *Sensors (Basel, Switzerland)*, 18(10):3521, 2018.
- [Zh19] Zhang, Tianfan; Li, Zhe; Deng, Zhenghong; Hu, Bin: Hybrid Data Fusion DBN for Intelligent Fault Diagnosis of Vehicle Reducers. *Sensors (Basel, Switzerland)*, 19(11):2504, 2019.
- [Zh20] Zhang, Y.; Beudaert, X.; Argandoña, J.; Ratchev, S.; Munoa, J.: A CPPS based on GBDT for predicting failure events in milling. *The International Journal of Advanced Manufacturing Technology*, 111(1-2):341–357, 2020.
- [Zh21a] Zhao, Qingsheng; Mu, Juwen; Han, Xiaoqing; Liang, Dingkan; Wang, Xuping: Evaluation Model of Operation State Based on Deep Learning for Smart Meter. *Energies*, 14(15):4674, 2021.
- [Zh21b] Zhou, Xuan; Ke, Ruimin; Yang, Hao; Liu, Chenxi: When Intelligent Transportation Systems Sensing Meets Edge Computing: Vision and Challenges. *Applied Sciences*, 11(20):9680, 2021.

- [Zh22] Zhang, Ning; Chen, Enping; Wu, Yukang; Guo, Baosu; Jiang, Zhanpeng; Wu, Fenghe: A novel hybrid model integrating residual structure and bi-directional long short-term memory network for tool wear monitoring. *The International Journal of Advanced Manufacturing Technology*, 120(9):6707–6722, 2022.



## Student Track



# Automated Statement Extraction from Press Briefings

Jüri Keller,<sup>1</sup> Meik Bittkowski,<sup>2</sup> Philipp Schaer<sup>3</sup>

**Abstract:** Scientific press briefings are a valuable information source. They consist of alternating expert speeches, questions from the audience and their answers. Therefore, they can contribute to scientific and fact-based media coverage. Even though press briefings are highly informative, extracting statements relevant to individual journalistic tasks is challenging and time-consuming. To support this task, an automated statement extraction system is proposed. Claims are used as the main feature to identify statements in press briefing transcripts. The statement extraction task is formulated as a four-step procedure. First, the press briefings are split into sentences and passages, then claim sentences are identified through sequence classification. Subsequently, topics are detected, and the sentences are filtered to improve the coherence and assess the length of the statements.

The results indicate that claim detection can be used to identify statements in press briefings. While many statements can be extracted automatically with this system, they are not always as coherent as needed to be understood without context and may need further review by knowledgeable persons.

**Keywords:** Computational Journalism; Claim Detection; Data Mining; Natural Language Processing

## 1 Introduction

Scientific press briefings are a valuable instrument in scientific communication. They consist of alternating expert speeches and answers to questions from the audience, which let scientists and journalists immediately and jointly address the information needs of journalists. The SMC press briefings<sup>4</sup> used in this work, typically begin with an introduction of the participating experts followed by moderated questions from journalists and answers from invited experts. Additionally, the key results are concluded in closing statements. Press briefings can directly contribute to a more fact-based media coverage by connecting journalists and scientists. Even though press briefings are highly informative, filtering this information and extracting relevant statements remains challenging and time-consuming due to the high entropy and domain-specific language used.

In this context, the research question will be answered: To what extent can claim detection be used to extract statements from scientific press briefings?

---

<sup>1</sup> Technische Hochschule Köln, Claudiusstraße 1, 50678 Köln, Germany jueri.keller@smail.th-koeln.de

<sup>2</sup> Science Media Center Germany, Schloss-Wolfsbrunnenweg 33, 69118 Heidelberg, Germany bittkowski@sciencemediacenter.de

<sup>3</sup> Technische Hochschule Köln, Claudiusstraße 1, 50678 Köln, Germany philipp.schaer@th-koeln.de

<sup>4</sup> [https://www.sciencemediacenter.de/alle-angebote/alle-angebote/?tx\\_solr%5Bfilter%5D%5B0%5D=type%3APress+Briefing](https://www.sciencemediacenter.de/alle-angebote/alle-angebote/?tx_solr%5Bfilter%5D%5B0%5D=type%3APress+Briefing)

To approach this domain-specific problem and automatically identify relevant statements from press briefings, a pipeline is proposed, relying on claims as a central element of a statement. A transformer based language model [CSM20] is used to classify claim sentences, which are subsequently filtered and clustered to improve the relevance and coherence of the emerging statements.

For example, a sentence like *"We do not yet know what a long covid course of Omicron infection looks like."* is considered a *complete claim*, while the sentence *"If I only have mild symptoms, it doesn't mean I won't have problems in the long run."* is only considered an *incomplete claim* as it can not be understood without context.

The main contribution of this work is the application of state-of-the-art methods in claim detection to develop a system that extracts statements from transcripts of scientific press briefings. Additionally, a novel dataset of 53 transcribed German press briefings is created and partially annotated. The dataset and implementation are made publicly available via GitHub<sup>56</sup> and Zenodo [Ke22].

## 2 Related Work

Automated extraction of statements from press briefings occurs at the intersection of journalism and computer science. While the premise and context emerge from journalism, the technical methods originate from computer science.

Various definitions of a claim exist [Da17; LEC16]. Generically, a claim indicates the author's position related to a main concept (topic) of a sentence [Le17]. Therefore, claims are indicated as the main feature of a statement. Generically, a claim is a short phrase people can have different opinions on [LEC16; LT16]. Since claims are conceptually complex, it is often difficult to determine what a claim is and what not [LT15]. To clarify, this work refers to a claim as a sentence asserting something on the main concept.

Claim detection is an information extraction task often part of a data processing pipeline solving more complex problems. Automatically detecting claims in texts has a variety of applications in fields like decision-making, argument mining, fact-checking, or document processing [Da17; Le18]. As diverse as the applications for claim detection are, so are the domains it is applied in, such as for example political, debate, legal, and the web [LT16].

Besides binary classification, different claim types or the part of the sentence that is a claim can be identified [Ko18; LT16]. Levy et al. [Le14] first formally defined the task of automated claim detection for computational argumentation while working on automatic argument mining [Le14; SI21]. The systems developed in the following years can be categorized by their ability to detect claims without further context or the scope of application, e.g., a

---

<sup>5</sup> Press Briefing Claim Dataset at GitHub: [https://github.com/jueri/press\\_briefing\\_claim\\_dataset](https://github.com/jueri/press_briefing_claim_dataset)

<sup>6</sup> Statement Extractor at GitHub: [https://github.com/jueri/statement\\_extractor](https://github.com/jueri/statement_extractor)

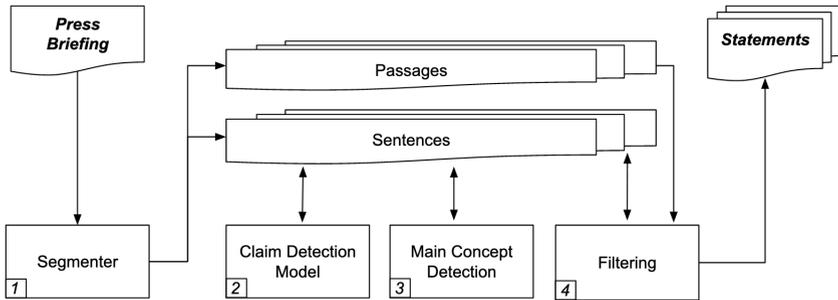


Fig. 1: Schematically visualization of the pipeline approach.

corpus of documents or a single one [Da17; LT15]. Chakrabarty et al. and Daxenberger et al. provide more extensive overviews of different claim detection models and datasets [CHM19; Da17].

### 3 Statement Detection System

Figure 1 visualizes the pipeline approach to extract statements from press briefings. The press briefing needs to be initially segmented into passages (1). Then, at the system’s core, a classification model is used to identify sentences containing claims (2). Further, the main concept of the press briefing and the sentences are detected (3). This information is used in the last step to filter out the sentences that are not or incomplete claims (4).

**1. Segmentation:** The press briefings are split by a sentence tokenizer, and neighboring and coherent sentences are combined into statements of multiple sentences. Therefore, the sentences are represented in vector space through spaCy [HMB20] and neighboring sentences are clustered by similarity with the TextSplit [AG15] algorithm. Starting with a list of vector representations of sentences, TextSplit calculates a score for each segment. By modifying the segment boundaries, the score changes. The optimal segmentation is determined by minimizing the aggregated segment scores [AG15; Mo20].

**2. Claim Detection:** A German language model [CSM20] based on the BERT [De19] architecture is fine-tuned for the task of sentence classification. The model’s hyperparameters are tuned to achieve the best results on available training data. The German BERT version GBERT is used as the foundational model for the classifier [CSM20]. This model is fine-tuned as a claim sentence classifier using a dataset of 3000 sentences created for this purpose from the publicly available SMC press briefings<sup>7</sup>. The dataset contains a total of 53 press briefings from a time span of over four years. It consists of 24,897 sentences with an average length of 17.31 tokens.

<sup>7</sup> <https://www.sciencemediacenter.de/alle-angebote>

**3. Main Concept Detection:** Based on the intuitive assertion that a claim sentence with a topic similar to the overall topic of the press briefing the sentence originates from is most relevant, the topics of both the sentences and the overall press briefings (i.e., the title and the introduction text) are detected. Therefore, Wikipedia articles are used as representations for the topic of a sentence or the whole press briefing. All Wikipedia articles related to tokens in the sentences and the title and the introduction texts are detected. The sentence is considered more relevant if a sentence can be linked to the same Wikipedia articles as the title or introduction text. For this task, the wikification APIs Dandelion<sup>8</sup> and TagMe<sup>9</sup> are used [FS10].

**4. Sentence Filtering:** Two techniques are applied to filter out claim sentences with a low topical similarity between the main concept of the sentence and the overall main concept to improve the coherence and relevance. The similarity is measured by two approaches based on embeddings or Wikipedia concepts. Both approaches create a similarity score that can be used for filtering with a minimum threshold.

The first approach calculates similarity by creating a vector representation from the title of the press briefing and the individual sentences and then measuring the cosine similarity between both. The vector representations are created by combining spaCy [HMB20] word embeddings. The second approach is based on the Wikipedia concepts detected as described previously. The confidence scores of shared Wikipedia concepts between the title or introduction text and the individual sentence are summed to create a score of topical relatedness.

## 4 Experimental Evaluation

Three experiments were conducted to evaluate the system's ability to extract statements from press briefings. Therefore, three new press briefings with 799 sentences from different categories were annotated as ground truth.

**Claim Detection:** For hyperparameter optimization, models with different configurations were trained for six epochs. The optimal number of epochs was determined in conjunction with the learning rate by comparing the evaluation loss rates after each epoch. Learning rates, ranging from 0.00005 to 0.01, were tested. The best results could be achieved with a learning rate of  $1e-5$  and three epochs. For the final system, a model with an F1 Score of 0.89, a precision of 0.92 and a recall of 0.86 was used. This model assigned slightly more false positive than false negative labels, but since the human in the loop can decide on the misclassified sentences, this behavior is preferred.

To evaluate the claim detection component of the system, claim sentences classified with minimal confidence scores of 0.7, 0.8 and 0.9 are analyzed independently. Since the

---

<sup>8</sup> <https://dandelion.eu/>

<sup>9</sup> <https://sobigdata.d4science.org/web/tagme/tagme-help>

Tab. 1: Results for detected *complete claims*. The first three columns hold the results for different confidence thresholds of the claim detection model. The last three columns contain the results of a system using a claim detection model threshold of 0.8 and the different main concept methods. The best results are highlighted in **bold**.

| Confidence | 0.9          | 0.8          | 0.7          | 0.8<br>embedding | 0.8<br>w. title | 0.8<br>w. intro |
|------------|--------------|--------------|--------------|------------------|-----------------|-----------------|
| F1         | 0.466        | <b>0.473</b> | 0.450        | <b>0.481</b>     | 0.339           | 0.341           |
| Precision  | <b>0.426</b> | 0.378        | 0.339        | <b>0.463</b>     | 0.456           | 0.430           |
| Recall     | 0.513        | 0.632        | <b>0.671</b> | <b>0.500</b>     | 0.270           | 0.283           |

confidence score limits the number of sentences considered as claims by the model, a score small enough to allow claim sentences to be classified and high enough to exclude non-claim sentences leads to the best results. The overall best results for detecting claims independent of their completeness were achieved at a confidence of 0.7 with an F1 score of 0.68, a precision of 0.89 and a recall of 0.55. By investigating individual results for the claim types, *complete claims* (that can stand on their own) and *incomplete claims* (that can not), it can be assessed if the filtering and sentence clustering methods improve the coherence. In total, a maximum of 167 of 224 *incomplete claims* and 102 of 152 *complete claims* claim sentences could be identified with various system configurations. With the claim detection model and minimum confidence of 0.8, the highest F1 score of 0.47 could be achieved. Raising the confidence threshold to 0.9 results in a higher precision of 0.43. Decreasing the score to 0.7 leads to a higher recall of 0.67.

**Statement Filtering:** Based on the results of the first experiments, the claim sentences with a confidence score of 0.8 are further filtered as this threshold achieved the best F1 score. To exclude incoherent statements, the similarity between the sentence’s main concept and the press briefing’s main concept is calculated. The main concepts of the press briefing based on *embeddings*, *title wikification* and *introduction wikification* are evaluated individually. For each method, a minimum similarity is chosen by investigating the similarity distribution for bends. With the *introduction wikification* based filtering, a precision of 0.43 was achieved. The *title wikification* method reaches a precision of 0.456, and the *embedding* method exceeds this score with the best precision of 0.463. All results are presented in table 1.

**Sentence Clustering:** The last experiment evaluates the sentence clustering method. Claim sentences with a confidence of at least 0.9 are enlarged to statements of multiple sentences. The statements are then assessed according to their coherence. To measure if the larger statements created by the similarity-based sentence clustering are more coherent, the *complete claim* and *incomplete claim* ratio for all methods are compared. With 53 % *complete claims*, the coherence of the statements extracted exceeds all other methods and the baseline. 46 % of the statements extracted with the claim detection module and a minimum confidence score of 0.9 are complete statements. The embedding method of the main concept module extracts 50 % *complete claims*.

## 5 Discussion and Conclusion

The analysis shows that the press briefings are very argumentative content full of claims. While most claims are incomplete and can only be understood with further information, 46 % can stand alone. The sentences extracted by the proposed system are highly likely to be claims, although not necessarily complete claims. This finding is supported by the low false-positive rate of the system considering all claims. Missing coherence is the main error of the statements extracted.

Compared to the initial results of the claim detection model with an accuracy of 0.89, the results achieved in the experiments are noticeably lower. This may be caused by the dataset used for training, which only contains *complete claims* and *no claims* and should be improved in future work. By adding a filter based on main concepts, some incoherent claim sentences could be successfully excluded, and the precision improved. A smaller confidence threshold of the initial claim detection module could improve these results further by providing more claim sentences in general. The embedding-based main concept method provided the best results. The Wikification-based methods come to an extent as some topics can only be expressed poorly by Wikipedia concepts, and other Wikipedia concepts may not exist or be captured by the model.

Assessing statements of multiple sentences, some statements previously classified as *incomplete claims* can be understood without additional information through the added neighboring sentences. By systematically adding sentences, more coherent statements could be created. However, this method sacrifices the conciseness of statements since they get longer.

In summary, this work investigates how statements can be mined from scientific press briefings. A dataset is created from publicly available press briefings and used to fine-tune a claim detection model. Additional methods are tested to improve the quality of the extracted statements. More context is added to the statements, and the statements are filtered based on their topic. The results are compared to a gold standard to evaluate the system's performance.

The results show that the system can differentiate between claims of any type and *no claim* sentences with high precision (0.89), but only half of the claims can be recalled (0.55). Considering only *complete claim* sentences, the system's overall performance decreases (F1 0.47), but relatively more complete claims can be identified. These results emphasize the system's difficulty differentiating between complete and incomplete claim sentences. The resulting statements gain coherence by adding more context. Similarly, incomplete claim sentences can be filtered out by low topical similarity, which improves the results slightly. The evaluations show that the presence of a claim can be used as an indicator to detect statements. However, the results leave room for improvement in many aspects. Especially the coherence of the extracted statements needs to be improved to extract statements that can stand on their own.

## References

- [AG15] Alemi, A. A.; Ginsparg, P.: Text segmentation based on semantic word embeddings. CoRR abs/1503.05543/, 2015, URL: <http://arxiv.org/abs/1503.05543>.
- [CHM19] Chakrabarty, T.; Hidey, C.; McKeown, K.: IMHO fine-tuning improves claim detection. In (Burstein, J.; Doran, C.; Solorio, T., eds.): Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, NAACL-HLT 2019, minneapolis, MN, USA, june 2-7, 2019, volume 1 (long and short papers). Association for Computational Linguistics, pp. 558–563, 2019, URL: <https://doi.org/10.18653/v1/n19-1054>.
- [CSM20] Chan, B.; Schweter, S.; Möller, T.: German’s next language model. In (Scott, D.; Bel, N.; Zong, C., eds.): Proceedings of the 28th international conference on computational linguistics, COLING 2020, barcelona, spain (online), december 8-13, 2020. International Committee on Computational Linguistics, pp. 6788–6796, 2020, URL: <https://doi.org/10.18653/v1/2020.coling-main.598>.
- [Da17] Daxenberger, J.; Eger, S.; Habernal, I.; Stab, C.; Gurevych, I.: What is the essence of a claim? Cross-domain claim identification. In (Palmer, M.; Hwa, R.; Riedel, S., eds.): Proceedings of the 2017 conference on empirical methods in natural language processing, EMNLP 2017, copenhagen, denmark, september 9-11, 2017. Association for Computational Linguistics, pp. 2055–2066, 2017, URL: <https://doi.org/10.18653/v1/d17-1218>.
- [De19] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In (Burstein, J.; Doran, C.; Solorio, T., eds.): Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, NAACL-HLT 2019, minneapolis, MN, USA, june 2-7, 2019, volume 1 (long and short papers). Association for Computational Linguistics, pp. 4171–4186, 2019, URL: <https://doi.org/10.18653/v1/n19-1423>.
- [FS10] Ferragina, P.; Scaiella, U.: TAGME: on-the-fly annotation of short text fragments (by wikipedia entities). In: Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10. ACM Press, Toronto, ON, Canada, p. 1625, 2010, ISBN: 978-1-4503-0099-5, URL: <http://portal.acm.org/citation.cfm?doid=1871437.1871689>, visited on: 10/07/2021.
- [HMB20] Honnibal, M.; Montani, I.; Boyd, S. V. L. A.: spaCy: Industrial-strength Natural Language Processing in Python, version 0.1, Feb. 2020, URL: <https://doi.org/10.5281/zenodo.1212303>.

- [Ke22] Keller, J.: Automated Statement Extraction from Press Briefings, version 0.1, Feb. 2022, URL: <https://doi.org/10.5281/zenodo.6047551>.
- [Ko18] Konstantinovskiy, L.; Price, O.; Babakar, M.; Zubiaga, A.: Towards automated factchecking: Developing an annotation schema and benchmark for consistent automated claim detection. CoRR abs/1809.08193/, 2018, URL: <http://arxiv.org/abs/1809.08193>.
- [Le14] Levy, R.; Bilu, Y.; Hershcovich, D.; Aharoni, E.; Slonim, N.: Context dependent claim detection. In (Hajic, J.; Tsujii, J., eds.): COLING 2014, 25th international conference on computational linguistics, proceedings of the conference: Technical papers, august 23-29, 2014, dublin, ireland. ACL, pp. 1489–1500, 2014, URL: <https://aclanthology.org/C14-1141/>.
- [Le17] Levy, R.; Gretz, S.; Sznajder, B.; Hummel, S.; Aharonov, R.; Slonim, N.: Unsupervised corpus-wide claim detection. In (Habernal, I.; Gurevych, I.; Ashley, K. D.; Cardie, C.; Green, N.; Litman, D. J.; Petasis, G.; Reed, C.; Slonim, N.; Walker, V. R., eds.): Proceedings of the 4th workshop on argument mining, ArgMining@EMNLP 2017, copenhagen, denmark, september 8, 2017. Association for Computational Linguistics, pp. 79–84, 2017, URL: <https://doi.org/10.18653/v1/w17-5110>.
- [Le18] Levy, R.; Bogin, B.; Gretz, S.; Aharonov, R.; Slonim, N.: Towards an argumentative content search engine using weak supervision. In (Bender, E. M.; Derczynski, L.; Isabelle, P., eds.): Proceedings of the 27th international conference on computational linguistics, COLING 2018, santa fe, new mexico, USA, august 20-26, 2018. Association for Computational Linguistics, pp. 2066–2081, 2018, URL: <https://aclanthology.org/C18-1176/>.
- [LEC16] Liebeck, M.; Esau, K.; Conrad, S.: What to do with an airport? Mining arguments in the german online participation project tempelhofer feld. In: Proceedings of the third workshop on argument mining, hosted by the 54th annual meeting of the association for computational linguistics, ArgMining@ACL 2016, august 12, berlin, germany. The Association for Computer Linguistics, 2016, URL: <https://doi.org/10.18653/v1/w16-2817>.
- [LT15] Lippi, M.; Torrioni, P.: Context-independent claim detection for argument mining. In (Yang, Q.; Wooldridge, M. J., eds.): Proceedings of the twenty-fourth international joint conference on artificial intelligence, IJCAI 2015, buenos aires, argentina, july 25-31, 2015. AAAI Press, pp. 185–191, 2015, URL: <http://ijcai.org/Abstract/15/033>.
- [LT16] Lippi, M.; Torrioni, P.: Argumentation mining: State of the art and emerging trends. ACM Trans. Internet Techn. 16/2, 10:1–10:25, 2016, URL: <https://doi.org/10.1145/2850417>.
- [Mo20] Mody, N.: Finding the best part of your podcast to promote via NLP, en, June 2020, URL: <https://towardsdatascience.com/finding-the-best-part-of-your-podcast-to-promote-via-nlp-f844a88b287a>, visited on: 09/08/2021.

- [SI21] Slonim, N.; Bilu, Y.; Alzate, C.; Bar-Haim, R.; Bogin, B.; Bonin, F.; Choshen, L.; Cohen-Karlik, E.; Dankin, L.; Edelstein, L.; Ein-Dor, L.; Friedman-Melamed, R.; Gavron, A.; Gera, A.; Gleize, M.; Gretz, S.; Gutfreund, D.; Halfon, A.; Hershovich, D.; Hoory, R.; Hou, Y.; Hummel, S.; Jacovi, M.; Jochim, C.; Kantor, Y.; Katz, Y.; Konopnicki, D.; Kons, Z.; Kotlerman, L.; Krieger, D.; Lahav, D.; Lavee, T.; Levy, R.; Liberman, N.; Mass, Y.; Menczel, A.; Mirkin, S.; Moshkovich, G.; Ofek-Koifman, S.; Orbach, M.; Rabinovich, E.; Rinott, R.; Shechtman, S.; Sheinwald, D.; Shnarch, E.; Shnayderman, I.; Soffer, A.; Spector, A.; Sznajder, B.; Toledo, A.; Toledo-Ronen, O.; Venezian, E.; Aharonov, R.: An autonomous debating system. en, *Nature* 591/7850, pp. 379–384, Mar. 2021, ISSN: 0028-0836, 1476-4687, URL: <http://www.nature.com/articles/s41586-021-03215-w>, visited on: 10/20/2021.



# MLProvCodeGen: A Tool for Provenance Data Input and Capture of Customizable Machine Learning Scripts

Tarek Al Mustafa<sup>13</sup>, Birgitta König-Ries<sup>123</sup>, Sheeba Samuel<sup>123</sup>

**Abstract:** Over the last decade Machine learning (ML) has dramatically changed the application of and research in computer science. With growing complexity, it becomes increasingly complicated to assure the transparency and reproducibility of advanced ML systems from raw data to deployment. In this paper, we describe an approach to supply users with an interface to specify a variety of parameters that together provide complete provenance information and automatically generate executable ML code from this information. We introduce *MLProvCodeGen* (Machine Learning Provenance Code Generator), a *JupyterLab* extension to generate custom code for ML experiments from user-defined metadata. ML workflows can be generated with different data settings, model parameters, methods, and training parameters and reproduce results in *Jupyter Notebooks*. We evaluated our approach with two ML applications, image and multiclass classification, and conducted a user evaluation.

**Keywords:** Provenance Management; Code Generation; Machine Learning; JupyterLab; Jupyter Notebooks; Reproducibility

## 1 Introduction

Machine Learning (ML) is the dominating data science approach today. ML solves various problems in many sectors. It also benefits the scientific community by supporting scientific workflows [De19] and database systems [Ma20; Va17]. ML workflows include steps to obtain results for given problems from raw data. These steps range from data preprocessing to deployment. Though they are common for every ML workflow, the specifics of the implementation, metadata of the entire experiment, and history of data points and sources used, differ for each ML model. Reproducibility of ML experiments, an increasingly important issue [Ba16; Hu18; SK21], can be enhanced by capturing this information as *provenance data*. We propose a method that allows users to generate code for ML pipelines by filling in templates with pre-defined parameters and variables. These templates incorporate all information needed for provenance tracking. We argue that this reduces the complexity of creating ML models while enhancing reproducibility. The main contributions of this work are: (1) define the minimum requirements to reproduce chosen ML workflows. (2) use these minimum requirements as a data model to build a template based system to automatically generate ML code in Jupyter notebooks<sup>4</sup> with multiple, user-chosen

---

<sup>1</sup> Heinz Nixdorf Chair for Distributed Information Systems, Jena, Germany

<sup>2</sup> Michael Stüfel Center Jena, Jena, Germany

<sup>3</sup> Friedrich Schiller University Jena {tarek.almustafa, birgitta.koenig-ries, sheeba.samuel}@uni-jena.de

<sup>4</sup> <https://jupyter-notebook.readthedocs.io/en/stable/>

parameters. (3) automatically capture and display provenance data from the generated notebooks to allow one-to-one reproductions by (4) inputting captured data into the system.

## 2 Related Work

**Provenance Data and Reproducibility.** Provenance plays a key role in reproducibility [Mi16]. Prospective provenance describes the specifications and steps that must be followed to generate a data product [Fr08]. Retrospective provenance captures what happened during the execution of a computational task. It is important that both provenance data types are captured and documented [De15; HDB17]. In our previous work, we investigated more factors that influence the reproducibility of ML experiments [SLK20].

**Provenance Data Models and Ontologies.** Provenance data models specify the format of metadata and which data points are represented. The *W3C PROV family of specifications* [MBC13] includes *The PROV Data Model (PROV-DM)* [Be13] and *The Provenance Ontology (PROV-O)* [Le13], an encoding of *PROV-DM* into *OWL2 Web Ontology Language*. Our previous work, the *REPRODUCE-ME Ontology* [SK17; SK18a], extends *PROV-O* and includes the *provenance-plan (P-PLAN)*<sup>5</sup> vocabulary to describe all computational and non-computational steps and data of scientific experiments in a machine-readable way.

**Provenance Capture Systems.** There have been a number of applications of these specifications and ontologies that may adopt or adjust existing data models. Other significant works include *PROV-ML* defined in [So19], which uses *W3C PROV* and *ML-Schema* to specify a provenance data model for complex tasks in the computational science and engineering domains and multiple systems that aim to capture provenance data automatically from either ML scripts [Na20; Sc17], model outputs [Ma17], computational notebooks [SK18b; SK20], specific workflow steps like data cleaning [PML20], or whole systems [Sc18].

**MLOps.** Systems applying DevOps practices to ML [Ta20] include AutoML<sup>6</sup>, MLflow [Za18], and ModelDB [Hi04]. They support ML development and deployment, including workflow management, data engineering, provenance management, and reproducibility. These systems target complex, custom-made ML products requiring contributions by several experts including data scientists and developers. In contrast, our work focuses on customizing predefined ML pipelines by lay users without the need for ML expertise.

**Code Generation and Templates.** Automatic code generation can increase productivity and consistency in ML scripts. Code generation tools can assist the development of automatic programming tools to improve programming productivity [LCB20]. However, supporting automatic code generation with multiple parameters raises complexity exponentially. *Train-Generator* provides and generates custom template code for ML<sup>7</sup>. It offers multiple options for preprocessing, model setup, training, and visualization. We build upon this system by developing a framework that can generate code for multiple ML tasks, generating executable notebooks, and integrating provenance data capture and visualization.

---

<sup>5</sup> <http://vocab.linkeddata.es/p-plan/version/13032014/>

<sup>6</sup> <https://cloud.google.com/automl/docs>

<sup>7</sup> <https://traingenerator.streamlit.app/>

### 3 MLProvCodeGen

In this section, we introduce *MLProvCodeGen*, a *JupyterLab* extension that explores how to support the reproducibility of ML experiments by combining template based code generation and provenance data capture, input, and visualization into one system. We implemented two example use cases: Image Classification and Multiclass Classification on tabular data, each with its set of customizable parameters. *MLProvCodeGen* was designed such that it can be extended to others. *MLProvCodeGen* is available online.<sup>8</sup>

Fig. 1 shows the system architecture consisting of a frontend plugin to capture information, and a backend plugin to process that information and generate notebooks from it.



Fig. 1: System Architecture of MLProvCodeGen

**Frontend.** The frontend plugin provides a user interface as shown in Fig. 2. Users can open

The screenshot shows the user interface for the MLProvCodeGen extension, divided into several sections:

- Data Ingestion:** Includes a dropdown for "Which data format do you want to use?" (set to "Public dataset"), a text input for "Select your dataset:" (set to "MNIST"), and a text input for "How many classes/output units?" (set to "10").
- Data Preparation:** A text input for "preprocessing:" with the value "Resize(256), CenterCrop(224), ToTensor(), grayscale to RGB".
- Data Segregation:** A note stating "Public datasets use premade testing datasets."
- Model Parameters:** Includes a checked checkbox for "Use GPU if available?", a dropdown for "Select a model:" (set to "resnet18"), a checkbox for "Do you want to use a pre-trained model?" (unchecked), a dropdown for "Optimizer" (set to "Adam"), a text input for "Learning rate" (set to "0.001"), and a dropdown for "Loss function" (set to "CrossEntropyLoss").

Fig. 2: Excerpt from Image Classification Input Elements in the User Interface

the extension by clicking the *MLProvCodeGen* button in the *other* section of *JupyterLab*'s home interface. At the bottom, users can submit their selected parameters to the system's backend. The user interface also allows users to input a provenance file from an experiment generated by *MLProvCodeGen* in the past in order to reproduce it.

**Backend.** The backend's main goal is to generate a notebook for either Image Classification or Multiclass Classification from user inputs. Each use case has a set of templates associated

<sup>8</sup> <https://mybinder.org/v2/gh/fusion-jena/MLProvCodeGen/main?urlpath=lab>

with it from which code can be generated. Therefore, the backend first selects a set of templates based on the specified use case, and then links variables from the user inputs to the templates. Since Jupyter Notebooks consist of cells, each cell is generated from a distinct template. Templates contain placeholder variables that are filled by the backend. For example, the template contains a placeholder called *dataset* and the backend extracts a *dataset* variable from the user inputs using `dataset = user_inputs['entity']['ex : DataIngestionData']['ex : dataset_id']` that is called *dataset* and has a value [`ex : dataset_id`]. When the templates are rendered, the value `ex : dataset_id` is written into the *dataset* placeholder and the output is appended to a notebook file. This way, a notebook file that was empty at the start is filled with rendered outputs from templates for all markdown and code cells. We use *Jinja*<sup>9</sup> as our templating language.

**Notebooks.** The notebooks are structured as follows: At the top is a markdown cell containing information about the ML task itself. The code cells below contain the installation command for the requirements and packages needed to run the notebook. Import statements are added directly after and provenance data capture is initialized. The remaining cells follow the structure of an ML pipeline. Each notebook has a cell for data ingestion, data preparation, data segregation, the model, training, and evaluation. At the bottom of each notebook are cells to generate a provenance graph, generate a provenance data file in JSON format, and cells to view the provenance data file and graph.

|                  |  |
|------------------|--|
| experiment_info  | creation_date, file_size, modification_date, task_type, title  |
| hardware_info    | CPU, GPUs, Operating_System, RAM   |
| packages         | All Python packages used in the notebook + the package version used  |
| notebook         | prov:type, creation_date, file_format, name, kernel, programming_language, programming_language_version  |
| data_ingestion   | start_time, end_time, execution_time, data_format, dataset_id, dataset_classes, feature_dimensions, dataset_description, root_location, training_samples, testing_samples, validation_samples        |
| data_preparation | start_time, end_time, execution_time, number_of_operations, operations   |
| data_segregation | start_time, end_time, execution_time, training_split, testing_split, validation_split  |
| model_parameters | start_time, end_time, execution_time, gpu_enable, pretrained, save_checkpoint, model_name, model_description, activation_function, output_neurons, loss_function, optimizer, optimizer_learning_rate |
| model_training   | start_time, end_time, execution_time, random_seed, resulting_model_seed, batch_size, epochs, print_progress  |
| model_evaluation | start_time, end_time, execution_time, evaluation_metrics(accuracy, loss, AUC, Confusion Matrix, F1, MAE, MSE)  |

Tab. 1: Provenance Data Model of MLProvCodeGen

**Provenance Data Capture.** All provenance information captured for notebooks generated by *MLProvCodeGen* is listed in Tab. 1. We capture provenance data using the *prov*<sup>10</sup> Python package. This allows us to specify entities, agents, and activities according to *PROV-DM*

<sup>9</sup> <https://jinja.palletsprojects.com/en/3.1.x/>

<sup>10</sup> <https://pypi.org/project/prov/>

specifications and build p-plans and collections adjacent to *PROV-O*. If a specific function was used to capture that information, *MLProvCodeGen* generates an activity describing it. Each code cell is an entity, has an activity that describes the execution of that cell, and a second entity that describes the data generated by the execution of that cell. Cell entities are ordered by specifying how a given cell was influenced by the ones executed before it. At the end of the notebook, the captured provenance data is saved to a JSON file and used to generate a provenance graph as seen in Fig. 3 and Fig. 4. A major downside of using the *prov* package is that the provenance capture has to be hard coded into the notebook at the time of notebook generation. This means that changes made by users after that point are only saved if users write them into the provenance data package themselves.

```

▶ ex:Cell Training:
▶ ex:Training Data:
▶ ex:Cell Evaluation:
▼ ex:Evaluation Data:
  ex:Accuracy: 0.933333373069763
  ex:Loss: 0.6712442636489868
  ex:Confusion Matrix: "[[13 1 0] [ 0 7 1] [ 0 0 8]]"
▼ ex:AUC:
  $: 0.99222
  type: "xsd:double"
  ex:F1 Score: "[0.96296296 0.875 0.94117647]"
  ex:Mean Absolute Error: 0.06666666666666667
  ex:Mean Squared Error: 0.06666666666666667
    
```

Fig. 3: Excerpt from a generated provenance JSON file in MLProvCodeGen



Fig. 4: Captured Evaluation Data in the Provenance Graph

**Provenance Data Input.** Any provenance data file generated by *MLProvCodeGen* can be uploaded to the system in the user interface to generate an identical reproduction of the code described by the provenance data. Uploaded files are processed by the backend in the exact same way as data input by users through the input elements in the user interface.

**Extensibility.** Due to the modular nature of *MLProvCodeGen*, users should have the ability to add new ML experiments to it. We have published step-by-step instructions in the online documentation. The different steps include: From an existing notebook (1) Write code

generation templates according to the notebook’s cells, (2) add provenance capture code to the templates following the data model and prior examples, (3) add new input elements to the user interface in line with the variables used in the templates, and (4) connect frontend and backend through a server call for the new ML experiment. Further evaluation would be necessary to assess the difficulty of extending *MLProvCodeGen*.

## 4 Preliminary Evaluation

We conducted a user evaluation to measure *MLProvCodeGen*’s user experience by combining an online survey via *LimeSurvey*<sup>11</sup> and a virtual installation of our program via *Binder*<sup>12</sup>. 12 entrants successfully completed the survey. All questions and completed answers are available online<sup>13</sup>. Our goal was to test the appropriateness and general usability of *MLProvCodeGen* for users from the computer science domain who may or may not be familiar with ML experiments, data provenance, and reproducibility. We asked users to self assess their level of proficiency with these terms, to complete hands-on user tasks, and consequently rate their experience using a variety of metrics. Of the 12 participants, eight answered the question regarding their professional background. All had a background in computer science or a related field. Prior knowledge about both machine learning and reproducibility was very mixed with all values from “poor” to “excellent” selected.

The key conclusions of the online survey are: (1) The explanations and instructions given are adequate to use *MLProvCodeGen* without outside help. (2) The user interface is intuitive and easy to use. (3) The generated notebooks have comprehensible structure and, depending on the users expertise, the code is coherent and understandable. (4) The provenance graph displays the provenance data as intended. However, for users without domain expertise, the graph is difficult to interpret. Due to its size, it is also challenging to find specific data points. Therefore, the provenance graph leaves room for improvement.

## 5 Conclusions and Future Work

In this paper, we presented *MLProvCodeGen*, a tool to support the reproducibility of machine learning experiments by combining template based code generation and provenance data capture, input, and visualization into one system. We evaluated our system by implementing two use case ML tasks, image classification and multiclass classification, and conducted a user evaluation. Future work on *MLProvCodeGen* includes improvements to the provenance graph, provenance data export, and adding more examples such as clustering. All source code, further information, explanations, a tutorial, the documented user evaluation, and an installation of *MLProvCodeGen* on a virtual machine are available online.<sup>14</sup>

---

<sup>11</sup> <https://www.limesurvey.org/>

<sup>12</sup> <https://mybinder.org/>, available at <https://mybinder.org/v2/gh/fusion-jena/MLProvCodeGen/main?urlpath=lab>

<sup>13</sup> <https://github.com/fusion-jena/MLProvCodeGen/tree/main/EvaluationResults>

<sup>14</sup> <https://github.com/fusion-jena/MLProvCodeGen>

## References

- [Ba16] Baker, M.: 1,500 scientists lift the lid on reproducibility. *Nature* 533/7604, 2016.
- [Be13] Belhajjame, K.; B'Far, R.; Cheney, J.; Coppens, S.; Cresswell, S.; Gil, Y.; Groth, P.; Klyne, G.; Lebo, T.; McCusker, J., et al.: Prov-dm: The prov data model. *W3C Recommendation 14/*, pp. 15–16, 2013.
- [De15] Dey, S.; Belhajjame, K.; Koop, D.; Raul, M.; Ludäscher, B.: Linking prospective and retrospective provenance in scripts. In: *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*. 2015.
- [De19] Deelman, E.; Mandal, A.; Jiang, M.; Sakellariou, R.: The role of machine learning in scientific workflows. *The International Journal of High Performance Computing Applications* 33/6, pp. 1128–1139, 2019.
- [Fr08] Freire, J.; Koop, D.; Santos, E.; Silva, C. T.: Provenance for computational tasks: A survey. *Computing in science & engineering* 10/3, pp. 11–21, 2008.
- [HDB17] Herschel, M.; Diestelkämper, R.; Ben Lahmar, H.: A survey on provenance: What for? What form? What from? *The VLDB Journal* 26/6, pp. 881–906, 2017.
- [Hi04] Hines, M. L.; Morse, T.; Migliore, M.; Carnevale, N. T.; Shepherd, G. M.: ModelDB: a database to support computational neuroscience. *Journal of computational neuroscience* 17/, pp. 7–11, 2004.
- [Hu18] Hutson, M.: Artificial intelligence faces reproducibility crisis. *Science* 359/6377, pp. 725–726, 2018.
- [LCB20] Le, T. H.; Chen, H.; Babar, M. A.: Deep learning for source code modeling and generation: Models, applications, and challenges. *ACM Computing Surveys (CSUR)* 53/3, pp. 1–38, 2020.
- [Le13] Lebo, T.; Sahoo, S.; McGuinness, D.; Belhajjame, K.; Cheney, J.; Corsar, D.; Garijo, D.; Soiland-Reyes, S.; Zednik, S.; Zhao, J.: Prov-o: The prov ontology. /, 2013.
- [Ma17] Ma, S.; Aafer, Y.; Xu, Z.; Lee, W.-C.; Zhai, J.; Liu, Y.; Zhang, X.: LAMP: data provenance for graph based machine learning algorithms through derivative computation. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. Pp. 786–797, 2017.
- [Ma20] Ma, L.; Ding, B.; Das, S.; Swaminathan, A.: Active learning for ML enhanced database systems. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Pp. 175–191, 2020.
- [MBC13] Missier, P.; Belhajjame, K.; Cheney, J.: The W3C PROV family of specifications for modelling provenance metadata. In: *Proceedings of the 16th International Conference on Extending Database Technology*. Pp. 773–776, 2013.

- [Mi16] Missier, P.: The lifecycle of provenance metadata and its associated challenges and opportunities. *Building Trust in Information/*, pp. 127–137, 2016.
- [Na20] Namaki, M. H.; Floratou, A.; Psallidas, F.; Krishnan, S.; Agrawal, A.; Wu, Y.; Zhu, Y.; Weimer, M.: Vamsa: Automated provenance tracking in data science scripts. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Pp. 1542–1551, 2020.
- [PML20] Parulian, N. N.; McPhillips, T. M.; Ludäscher, B.: A model and system for querying provenance from data cleaning workflows. In: *Provenance and Annotation of Data and Processes*. Springer, pp. 183–197, 2020.
- [Sc17] Schelter, S.; Boese, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: *Machine Learning Systems Workshop at NIPS*. Pp. 27–29, 2017.
- [Sc18] Schelter, S.; Böse, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Declarative metadata management: A missing piece in end-to-end machine learning. *Proceedings of SYSML 18/*, 2018.
- [SK17] Samuel, S.; König-Ries, B.: REPRODUCE-ME: ontology-based data access for reproducibility of microscopy experiments. In: *European Semantic Web Conference*. Springer, pp. 17–20, 2017.
- [SK18a] Samuel, S.; König-Ries, B.: Combining P-Plan and the REPRODUCE-ME ontology to achieve semantic enrichment of scientific experiments using interactive notebooks. In: *European semantic web conference*. Springer, pp. 126–130, 2018.
- [SK18b] Samuel, S.; König-Ries, B.: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In: *ISWC (P&D/Industry/BlueSky)*. 2018.
- [SK20] Samuel, S.; König-Ries, B.: Reproducemegit: a visualization tool for analyzing reproducibility of jupyter notebooks. In: *Provenance and Annotation of Data and Processes*. Springer, pp. 201–206, 2020.
- [SK21] Samuel, S.; König-Ries, B.: Understanding experiments and research practices for reproducibility: an exploratory study. *PeerJ* 9/, e11140, 2021.
- [SLK20] Samuel, S.; Löffler, F.; König-Ries, B.: Machine learning pipelines: provenance, reproducibility and FAIR data principles. In: *Provenance and Annotation of Data and Processes*. Springer, pp. 226–230, 2020.
- [So19] Souza, R.; Azevedo, L.; Lourenço, V.; Soares, E.; Thiago, R.; Brandão, R.; Civitarese, D.; Brazil, E.; Moreno, M.; Valduriez, P., et al.: Provenance data in the machine learning lifecycle in computational science and engineering. In: *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, pp. 1–10, 2019.

- [Ta20] Tamburri, D. A.: Sustainable mlops: Trends and challenges. In: 2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC). IEEE, pp. 17–23, 2020.
- [Va17] Van Aken, D.; Pavlo, A.; Gordon, G. J.; Zhang, B.: Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM international conference on management of data. Pp. 1009–1024, 2017.
- [Za18] Zaharia, M.; Chen, A.; Davidson, A.; Ghodsi, A.; Hong, S. A.; Konwinski, A.; Murching, S.; Nykodym, T.; Ogilvie, P.; Parkhe, M., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. 41/4, pp. 39–45, 2018.



# To Iterate Is Human, to Recurse Is Divine — Mapping Iterative Python to Recursive SQL

Tim Fischer<sup>1</sup>

**Abstract:** Writing complex algorithms and iterative computations in SQL is difficult at best, commonly leading to code that intermingles looping control flow with database access. This yields programs with control flow that rapidly hops in and out of the database, with each roundtrip incurring significant overhead. We present the ByePy compiler, which can compile entire Python functions directly to plain recursive SQL:1999 queries. By doing so, the compilation eliminates all but a single roundtrip, leading to runtime speedups of up to an order of magnitude.

**Keywords:** SQL; Python; Compilation

## 1 Introduction

The performance of all applications stands and falls with the efficient use of resources, e.g., compute, memory, network, etc. In the realm of database-backed applications, developers can optimize the usage of many, if not most, of these resources by adhering to the decades-old mantra of database development: “*Move your computation close to the data*” [RS87]. To do so, developers need to express their computations *and* data in a form that databases can ingest and process. Nowadays, this usually means storing the data in tables and expressing the computations over it in terms of the ubiquitous SQL.

Following the mantra and moving *all* computation into the database is often difficult. The main stumbling block is the *impedance mismatch* between the *declarative* paradigm underlying SQL and the *imperative* paradigm most developers are more familiar with. This mismatch leads developers most comfortable with imperative programming to write programs that perform the bulk of their computation outside the database, i.e., far away from the data. Such programs are littered with intermittent database access throughout; consider the implementation of function `march` in the left half of Fig. 1 (`march` computes the outline of a 2D shape). During the execution of such a program, it will have to perform many complete round trips 🔄, i.e., the program’s control flow has to move from Python to the database 🗄️ and back again 🏠. With each additional round trip incurring resource overhead.

Intending to reach code that behaves like the right side of Fig. 1, that is, minimizing round trips 🔄 while retaining the imperative paradigm, Ramachandra et al. introduced Froid

---

<sup>1</sup> Eberhard Karls Universität Tübingen, Wilhelm Schickard Institut [tim.fischer@uni-tuebingen.de](mailto:tim.fischer@uni-tuebingen.de)

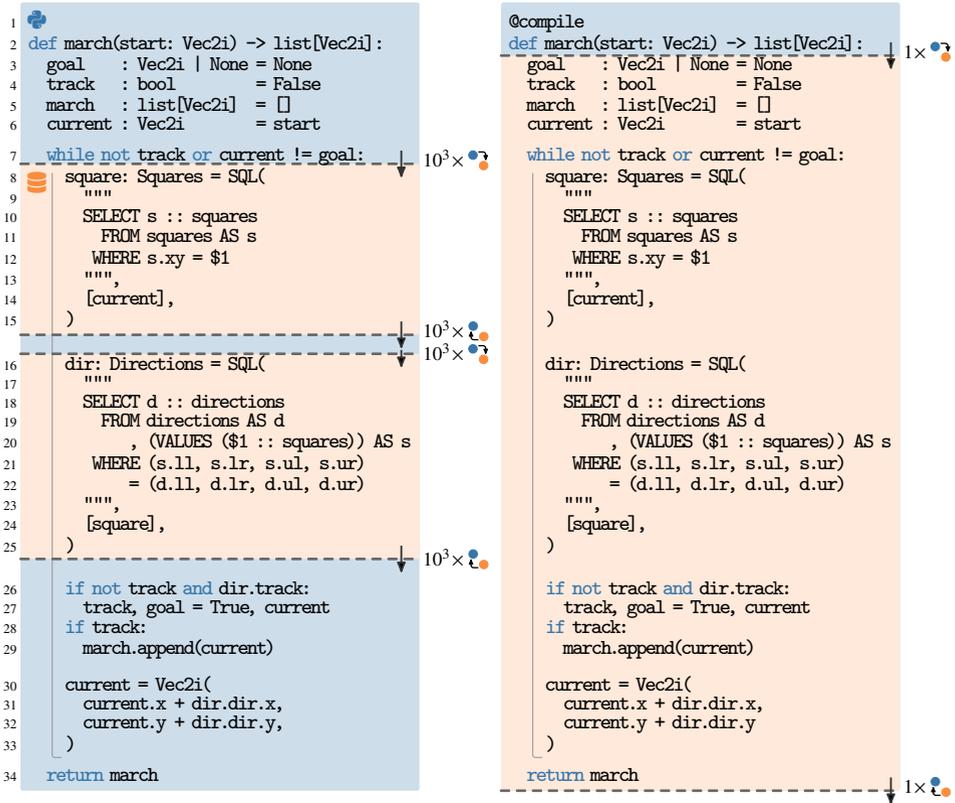


Fig. 1: Execution context boundaries between Python and the database in iterative code with embedded queries and code compiled with ByePy.

[Ra17]. Focussing on optimizing simple non-iterative PL/SQL functions by compiling them to SQL, Froid was restricted to code with linear control flow. Building on this idea, Hirn and Grust introduced a new compilation approach that extended the compilation to support non-linear control flow constructs, e.g., loops [HDG20; HG20; HG21]. In [FHG22], we demonstrated the applicability of this approach to languages other than PL/SQL via ByePy, a Python frontend for [HG21]. Since then, we have been working on extending the set of language constructs that ByePy can digest. In addition to the features presented in [FHG22], it now also supports the following:

- dictionaries with string-valued keys and “JSON-valued” entries,
- delete statements on containers (e.g., `del some_list[2:5]`),
- falsifiability of builtins (e.g. `while some_list: ...`),
- nested `None` disambiguation (e.g. `if ... and x is not None and ...: ...`), and
- arbitrarily nested (augmented-)assignment (e.g., `v[0].my_dict["key"] += 1`).

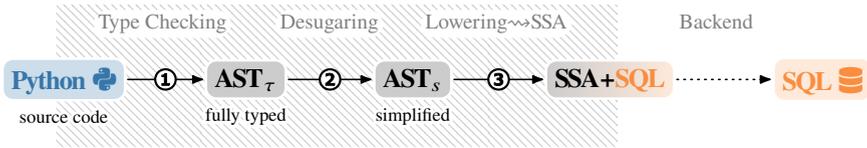


Fig. 2: Intermediate representations in the ByePy frontend.

## 2 The ByePy Compiler

The ByePy compiler consists of two major parts. The first of which is the novel frontend which compiles Python to a mix of Static Single Assignment form (SSA) to represent the general control flow and SQL to represent embedded expressions. Furthermore, the second part consists of the SSA to SQL compilation pipeline elaborated in [HG21].

ByePy focuses on computations over database-resident data; as such, we limit the supported language features to a subset most commonly used in conjunction with such computation—think conditionals, loops, flow control statements like `break`, and complex assignments like `v[0].att += 1`. The frontend wrangles Python programs using this subset into the SSA+SQL representation in three distinct stages, as depicted in Fig. 2, those being ① performing soundness and type checking, ② simplifying particularly complex statements and expressions, and ③ lowering the AST into the combined SSA+SQL representation.

**① Type Checking** Aside from the *conceptual* impedance mismatch between the imperative paradigm of Python and the declarative paradigm of SQL there is a *structural* impedance mismatch. Python is a dynamically typed language, meaning types are reified *at runtime*. SQL, on the other hand, is statically typed, where types are already explicitly declared *before runtime*. To bridge this gap, ByePy implements a type-checking stage that enriches a minimally typed AST such that each expression is annotated with an appropriate type. Minimally typed refers to the fact that ByePy requires type annotations in situations where the type inference does not have enough information—e.g., function parameters, variable declarations, or function return types.

**② Desugaring** Following the type checking, we rewrite parts of the AST in terms of simpler syntactic constructs. Doing so simplifies the subsequent steps greatly as it limits the amount of different syntactic constructs they are required to handle. These rewrites include the following:

- reducing the set of used operators by rewriting more complicated ones in terms of simpler one (e.g., `x not in y`  $\mapsto$  `not (x in y)`),
- placing appropriate “casting expression” where Python’s duck-typing would do so during runtime (e.g., `if some_list: ...`  $\mapsto$  `if len(some_list) > 0: ...`),
- replacing stateful expressions with an equivalent series of assignments and variables, and
- rewriting of arbitrarily complex assignments into their simplest equivalents (e.g., `v[0]["key"] += 1`  $\mapsto$  `v = [{**v[0], "key": v[0]["key"] + 1}] + v[1:]`).

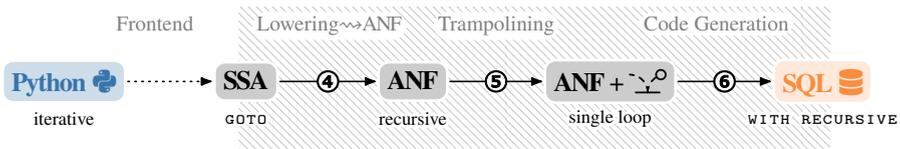


Fig. 3: Intermediate representations in the ByePy backend.

③ **Lowering**  $\rightsquigarrow$  SSA The desugaring produces an equivalent AST in which all *stateful computation* is expressed solely through *single variable assignments* and *statement-level control flow*. With such an AST in hand, we can finally apply the lowering. In short, we translate all control flow constructs into equivalent labeled blocks and GOTOs while translating the remainder directly to equivalent SQL queries.

Once the frontend has compiled all Python specifics away, the backend pipeline designed by Hirn and Grust comes into play. In short, it applies a series of three transformations depicted in Fig. 3 which results in an equivalent *recursive SQL*:1999 query, i.e., a recursive common table expression (CTE).

④ **Lowering**  $\rightsquigarrow$  ANF The control flow, which is expressed in terms of SSA, is lowered to Administrative Normal Form (ANF) using a transformation described by Chakravarty et al. in [CKZ04]. In short, we turn all blocks into functions, all GOTOs into calls of those functions, and all assignments into LET-expressions. Of particular note is that the lowering to ANF places the calls replacing the GOTOs in the tail position.

⑤ **Trampolining** Lowering to ANF, generally, leads to a family of recursive functions. To facilitate the compilation into the SQL-based CTE form, we subject this family of functions to the trampoline transformation [GFW99], which yields a single-loop computation that fits the CTE semantics.

⑥ **Code Generation** The last step is to generate SQL code equivalent to the program in trampolined ANF. We can do so by encoding LET-expressions as LATERAL-joins, recursive calls as SELECT-clauses containing the parameters, and conditional expressions as UNIONS of the individual branches with mutually exclusive WHERE-clauses.

The generated query implements the trampoline through a recursive CTE in which each recursive step handles one trampoline transition. Thus, all program state resides within the working table; this includes both the state of the control flow and the bindings of the program’s live variables. Each recursive step generates a new row representing the result of the transition, containing both new variable bindings and copies of the unchanged bindings. This behavior can lead to performance impacts for functions whose local variables carry sizable data structures—like long arrays. The right edge of Fig. 4a exemplifies this.

Tab. 1: A collection of Python functions with roundtrips before and speedup after compilation.

| Function   | CC | Loops         | # 🔄<br>per call | Runtime (Speedup)<br>after compilation |
|--|----|---------------|-----------------|--|
| <code>march</code> track border of 2D object (Marching Squares)                | 5  | 📄             | 2000            | 13% ( 7.6×)                            |
| <code>savings</code> optimize supply chain of a TPC-H order                    | 4  | qq, 📄, qq, qq | 18              | 5% (19.5×)                             |
| <code>packing</code> pack TPC-H lineitems tightly into containers              | 9  | qq, 📄, 📄      | 45              | 16% ( 6.3×)                            |
| <code>force</code> <i>n</i> -body simulation (Barnes-Hut quad tree)            | 5  | q, 📄          | 126             | 27% ( 3.9×)                            |
| <code>margin</code> buy/sell TPC-H orders to maximize margin                   | 5  | q, 📄, 📄       | 61              | 24% ( 4.2×)                            |
| <code>markov</code> Markov-chain based robot control                           | 5  | 📄, qq, qq     | 3000            | 39% ( 2.6×)                            |
| <code>vm-collatz</code> calculate the <i>collatz conjecture</i> on a simple VM | 17 | 📄             | 67              | 30% ( 3.3×)                            |
| <code>vm-padovan</code> calculate the <i>padovan sequence</i> on a simple VM   | 17 | 📄             | 7100            | 12% ( 8.5×)                            |

### 3 Experimenting with the *Divine*

We claim that compiling Python to SQL using the ByePy pipeline has the capability of speeding up programs quite drastically depending on their complexity. This section supports this claim through eight sample functions with varying complexity and quantifies how the compilation affects their runtimes. We performed all measurements with PostgreSQL 11.3 and Python 3.8 running on a 64-bit Linux x86 host (2× AMD EPYC™ CPUs at 2.8 GHz, 2 TB of DDR4 RAM). All presented results represent the median of five runs.

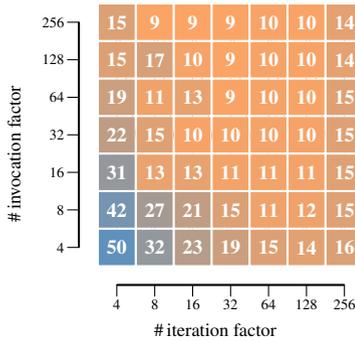
Tab. 1 lists the eight sample functions; you can find the original Python source code, compiled SQL queries, and appropriate data generators on *GitHub*<sup>2</sup>. They cover a wide range of use cases and classes algorithms, e.g., optimization problems over TPC-H data, simulation of VMs, or algorithms over 2D point data. The columns **CC** (cyclomatic complexity) and **Loops** give some insight into the structure of the functions without looking at the source code. Especially the latter gives a sense of where the embedded queries (q) sit inside the functions control flow.

Zooming in on `march` in Fig. 4a, we can see that the performance of the compiled functions can be sensitive to the size of the data they operate on. In the lower-left corner, the planning required for the query outweighs to performance benefits introduced by the compilation. Fig. 4b shows us the expected effect on the number of roundtrips; increasing the number of iterations and invocations also increases the round trips a program encounters. Furthermore, we can also see that compilation significantly decreases the required round trips 🔄.

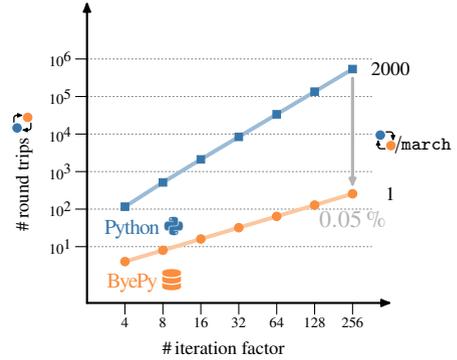
### 4 Wrapping Up

When working with database-resident data, most Python developers opt to perform complex computations outside of the database. Embedding database access in (potentially deeply nested) loops raises significant performance concerns. To minimize the resulting round

<sup>2</sup> <https://github.com/ByePy/examples>



(a) Runtime (in % of Python) after compilation



(b) Comparison of round trips along the diagonal of Fig. 4a

Fig. 4: Deeper analysis of the runtimes and round trips of the `march` function.

trips such code experiences during runtime, we developed ByePy; a Python frontend for the PL/SQL to SQL compilation pipeline described in [HG21]. The compilation yields runtime speedups of up to an order of magnitude on a wide range of functions, from optimization problems to stochastic processes.

Currently, we are in the process of introducing PostgreSQL’s geometric types, functions, and operators as optional extensions to the ByePy dialect. Beyond these extensions, we plan to expand the set of Python language features ByePy supports. Extensions that appear to be in immediate reach include things like multi-variable assignments (e.g., `a, b = 1, 2`) and comprehensions (e.g., `[f(e) for e in some_list if p(e)]`). In the *not so near* future, we hope to integrate the compilation directly into the `@compile` decorator, enabling easier use of ByePy in the wild.

## References

- [CKZ04] Chakravarty, M. M.; Keller, G.; Zadarnowski, P.: A Functional Perspective on SSA Optimisation Algorithms. COCV’03/, 2004.
- [FHG22] Fischer, T.; Hirn, D.; Grust, T.: Snakes on a Plan: Compiling Python Functions into Plain SQL Queries. In. SIGMOD ’22, 2022.
- [GFW99] Ganz, S. E.; Friedman, D. P.; Wand, M.: Trampoline Style. In. ICFP ’99, 1999.
- [HDG20] Hirn, D.; Duta, C.; Grust, T.: Compiling PL/SQL Away. In. CIDR ’20, 2020.
- [HG20] Hirn, D.; Grust, T.: PL/SQL Without the PL. In. SIGMOD ’20, 2020.
- [HG21] Hirn, D.; Grust, T.: One WITH RECURSIVE is Worth Many GOTOs. In. SIGMOD ’21, 2021.
- [Ra17] Ramachandra, K.; Park, K.; Emani, V.; Halverson, A.; Galindo-Legaria, C.; Cunningham, C.: Froid: Optimization of Imperative Programs in a Relational Database. In. VLDB ’17, 2017.
- [RS87] Rowe, L.; Stonebraker, M.: The POSTGRES Data Model. In. VLDB ’87, 1987.

# Optimizing Query Processing in PostgreSQL Through Learned Optimizer Hints

Jerome Thiessat,<sup>1</sup> Lucas Woltmann,<sup>1</sup> Claudio Hartmann,<sup>1</sup> Dirk Habich<sup>1</sup>

## Abstract:

Query optimization in database systems is a crucial issue and despite decades of research, it is still far from being solved. Nowadays, query optimizers usually provide hints to be able to steer the optimization on a query-by-query basis. However, setting the best-fitting optimizer hints is challenging. To tackle that, we present a learning-based approach to predict the best-fitting hints for each incoming query. In particular, our learning approach is based on simple gradient boosting, where we learn one model per query context for fine-grained predictions rather than a single global context-agnostic model as proposed in related work. We demonstrate the efficiency as well as effectiveness of our learning-based approach using the open-source database system PostgreSQL and show that our approach outperforms related work in that context.

**Keywords:** Query Optimization; Hint Set Prediction; Gradient Boosting

## 1 Introduction

Every database system features a query compiler that converts each incoming declarative SQL query into a query execution plan (QEP). The most important component of such a query compiler is the query optimizer. The task of this optimizer is to determine the most efficient QEP. Despite decades of research activities, query optimization is still far from being solved [Le15]. According to [Ch98], the most challenging issues for the optimization of complex SQL queries are: (i) finding a good join order and (ii) selecting the best-fitting physical join implementation for each join within the chosen join order. To solve these challenges, a traditional query optimizer uses three components: the enumerator which spans – according to the relational algebra – the search space of all possible QEPs, the cost model to assess the cost of any given QEP prior to its execution, and the cardinality estimator which delivers the size of intermediate results and base tables as most crucial input to the cost model.

Such a traditional query optimizer can be found in open-source database systems, e.g., PostgreSQL [Po]. However, a disadvantage of this traditional optimizer approach is that the determined QEP for a query can vary widely in quality [Le15]. The quality variance possibly originates from miss-predicted intermediate results from PostgreSQL's cost estimator and

---

<sup>1</sup> Technische Universität Dresden, Dresden Database Research Group, 01062 Dresden, Germany,  
{jerome.thiessat,lucas.woltmann,claudio.hartmann,dirk.habich}@tu-dresden.de

|                 | PostgreSQL w/ default hints | PostgreSQL w/ <b>our learned hint approach</b> |
|-----------------|-----------------------------|--|
| Stack-Benchmark | 5,445.78 sec                | 2,802.54 sec                                   |

Tab. 1: Workload execution times of the real-world Stack benchmark [Ma22] with default and our learned optimal physical operator hints (PostgreSQL v14.2). More details in Section 4.

PostgreSQL falling back to a genetic optimizer upon surpassing a certain amount of joins in a query. To overcome this issue, PostgreSQL provides a set of well-defined optimizer hints to steer the optimizer on a query-by-query basis. For instance, the usage of the physical join operator `hash join` can be enabled or disabled using a specific hint. In general, PostgreSQL features six Boolean hints for physical operators; three for joins and three for scans. In the default setting, all six physical operator hints are activated to allow the optimizer’s enumerator to span the largest possible search space.

To show the significance of hinting, Table 1 compares the workload execution times of the real-world Stack benchmark [Ma22] for PostgreSQL using (i) the default hint setting and (ii) our *learned hint approach* for the six physical optimizer hints. As shown, the utilization of our *learned hints* dramatically reduces the workload execution time. We achieve this benefit by learning a simple gradient boosting model per query context for fine-grained hint predictions. Moreover, our *learning component* is intentionally designed as a separate loosely-coupled component for PostgreSQL to guarantee broad applicability for different PostgreSQL versions. To sum up, our contributions in this paper are:

- In Section 2, we introduce preliminaries and describe the related work in this context.
- Based on these considerations, we describe the key features of our *context-aware learning approach* for hint sets in Section 3.
- Then, we present selected evaluation results to show the efficiency and effectiveness of our approach compared to state-of-the-art and related work in Section 4.

Finally, we conclude our findings with a short summary in Section 5.

## 2 Preliminaries and Related Work

Fundamental for our contribution is the procedure of hinting the PostgreSQL query compiler. PostgreSQL offers various hints as planner method options<sup>2</sup>. In the following, we use the terminology *hint* as PostgreSQL’s planner methods options, *hint set* as a combination of hints, and *hinting* as the procedure of setting the planner method options in PostgreSQL accordingly. Hinting in PostgreSQL is syntactically a trivial task and can be realized by e.g., `set enable_hashjoin = false`; to disable hash joins as a prefix annotation to an SQL query. As already stated in Section 1, we focus on the six Boolean hints that are considered by PostgreSQL’s planner<sup>3</sup> for physical operators. These hints consider three joins, i.e., *hash*,

<sup>2</sup> <https://www.postgresql.org/docs/current/runtime-config-query>

<sup>3</sup> <https://www.postgresql.org/docs/14/planner-optimizer.html>

*nested-loop*, and *merge join* as well as three scan operations, i.e., *index*, *sequential*, and *index-only scan*. These hints only allow to enable or disable the corresponding physical operators which however influence the whole optimization procedure.

As clearly demonstrated in [Ma22], these six physical operator hints can be efficiently used to steer the query optimization to produce more efficient QEPs, but hinting is a challenging task in general. To tackle that challenge, [Ma22] proposed a learning-based approach called BAO – the bandit optimizer, which is the most relevant related work for our approach. From a high-level perspective, BAO learns a mapping between an incoming query and the optimizer hints the query optimizer should use for that query using reinforcement learning. For that, BAO treats each hint set as an arm in a contextual multi-armed bandit problem and learns a single model that predicts which hints will provide the best run-time for an incoming query. In general, BAO works as follows: For every SQL query, the underlying PostgreSQL query optimizer produces  $n$  QEPs; one for each hint set. Afterwards, each QEP is transformed into a vector tree and the resulting vector trees are fed into a tree convolutional neural network (BAO’s single model) predicting the execution time of each QEP. The QEP with the least predicted execution time is finally selected for execution. Once the QEP is executed, the selected QEP and the real execution time is added to BAO’s experience. These experiences are used to periodically retrain the single model.

To the best of our knowledge, BAO is the only work that relates closely to our challenge of predicting hint sets for incoming queries. However, BAO has the following shortcomings. Firstly, BAO uses a single *global* model across all incoming queries to predict hint sets. This does not allow for fine-grained nuanced predictions for queries that differ only marginally, for example in predicates. Secondly, BAO predicts hint sets *indirectly* by predicting execution times and then inferring on the best hint set afterward. This indirection step is not necessarily beneficial, as multiple hint combinations need to be evaluated during query optimization time to determine the best-performing QEP. Lastly, BAO only investigates a reduced amount of hint sets to keep the necessary additional effort during query optimization time as low as possible. That means, for the six physical operator hints, there are  $2^6 = 64$  possible hint sets, but only 25 are considered in BAO. In these 25 hint sets, the globally optimal solution might not even be included.

### 3 Context-Aware Hinting

To overcome the above presented shortcomings of BAO, we introduce a novel *learning-based approach* called *POSGB* to predict the best-fitting hints for each incoming query in this section. The key features of *POSGB* are: Firstly, we deploy context-sensitive models, where we build a learned model for each set of joined tables of a workload. The idea behind this is that each set of joined tables represents a self-contained context, since the queries per context are thus reasonably homogeneous with respect to the joins and differ only in the filter predicates. Using this divide-and-conquer approach allows us to predict on a fine-grained basis, where BAO uses a context-agnostic approach. Secondly, within each

context, our models follow supervised classification. This means that we directly predict a hint set from a query, rather than inferring indirectly on multiple QEPs with estimated costs. By relying on a classification task, we are also able to consider the whole search space of the  $2^6$  possible hint sets rather than a reduced subset. Lastly, to reduce the additional effort of using these models during query optimization time, we utilize a classical gradient boosting model within each context, rather than one single global neural network. Gradient boosting models are a learning method using a sequential ensemble of smaller models (i.e., weak base learners) typically trained using momentum based optimization (e.g., gradient descent). Based on that, for each query during query optimization time, a context classification and a prediction with one small simple model has to be conducted. Naturally, these learned context models traverse a training phase before being able to predict an incoming query.

**Training Phase:** *POSGB*'s training phase follows the same principle for every context. Within each context, a set of input queries is first featurized query compiler independently (i.e., not relying on QEPs) in the fashion of [Ki19; Wo19] by encoding filter predicates. Moreover, since we deploy supervised learning, we also label each training query with the optimal hint set by exhaustively evaluating each query-hint-set combination. Notably, we also investigated the use of PostgreSQL's *EXPLAIN* functionality to reduce our labeling effort to a minimum. However, evaluating the Pearson correlation coefficient between guessed and real cost of an appropriate sample did not result in any notable correlation. Since we could not even observe any relation in the order of best-to-worst hint set, we deemed *EXPLAIN* not suitable for our task. This requires the execution of all training queries with all hint sets. Then, the hint set that produces the least execution time is determined and used as a label for each query.

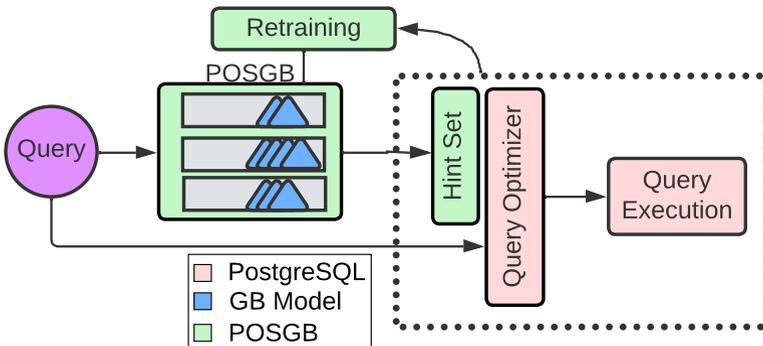


Fig. 1: Workflow of *POSGB*

**Model Inference:** *POSGB*'s query inference follows Figure 1. Firstly, the optimal hint set of an incoming query is predicted by *POSGB*. There, the query is assigned to a context and featurized analogously to the training phase. *POSGB* then predicts the label – a hint set – from the featurized input query. Secondly, the query with the predicted hint set is propagated to PostgreSQL for query optimization and execution.

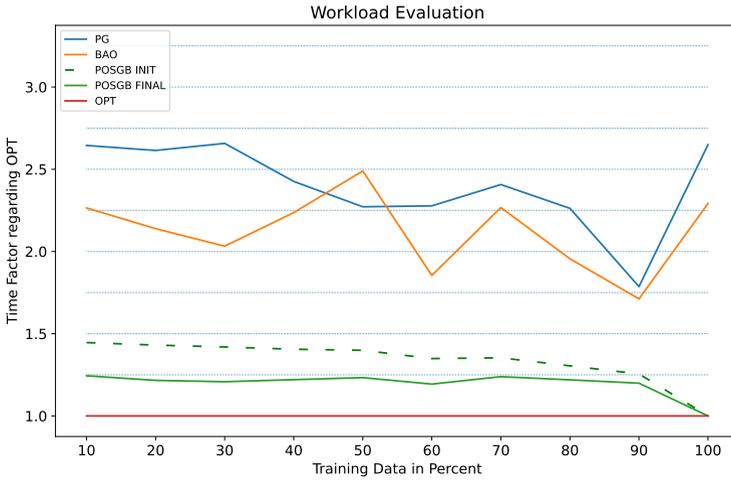


Fig. 2: Workload Evaluation of Multiple Data Splits

An important challenge in hint set prediction and generally learned query optimization, is handling unseen, badly performing queries. Generally, our model does not support detection of unusually long executing queries. For this reason, we provide a model adaptation by detecting such queries. Each query is executed with a timeout (i.e., statement timeout in PostgreSQL), where the timeout is context-sensitive and based on already executed queries within the specific context. The longest running seen query per context defines the timeout. By doing so, any query that exceeds the specified context-sensitive timeout threshold is considered as critical. On the one hand, such critical queries are canceled and re-executed with the PostgreSQL default hints. On the other hand, the critical queries are exhaustively evaluated in an asynchronous manner to determine the optimal hint set. Based on this new experience, an updated model for the specific context is trained. Upon having the newly trained model ready, the old model is exchanged by the new one.

## 4 Evaluation

To show the efficiency as well as effectiveness of *POSGB*, we conducted a comprehensive evaluation on a machine with an Intel Xeon Gold 6126 CPU, an ASPEED Graphics Family GPU, and 95 GiB memory. Our whole evaluation is based on the Stack benchmark, consisting of 100 GB data as well as 6191 queries from real-world examples [Ma22]. We evaluated four different scenarios: (i) PostgreSQL native with default hint setting (PG), (ii) BAO [Ma22], (iii) the initial evaluating of *POSGB* (INIT), and (iv) after retraining has been deployed (FINAL). Figure 2 shows the most important result. Depicted are training splits on the x-axis and the relative workload time factor regarding the global optimal solution on the y-axis. We determined the global optimal solution by an exhaustive search over all

queries and all hint sets. Important to note is that the Stack queries are classified into eleven contexts and that the training splits are fully random. Additionally, the *100%* split marks representative learning, which uses all data for training and testing. Notably, every but the *100%* split are vaulted (i.e., test queries are not seen by the model).

The most important results can be summarized as follows. Firstly, we observe that *POSGB* dramatically reduces the workload execution times for all training splits compared to PostgreSQL native as well as to the most related approach BAO [Ma22]. In particular, the workload times using *POSGB* are much closer to the global optimal solution. Secondly, with more training data, the workload times are continuously reduced, which is not the case with BAO as already shown in [He22] due to catastrophic forgetting. Moreover, *POSGB* performs well even for the small splits like *10%*. Notably, this performance comes with a caveat as each query has to be labeled. This sums up to roughly *12h* for the *10%* split. However, we deem this time still feasible as it is not impractical and does not interfere with the model's on-line behavior. Furthermore, representative learning shows our model is capable of learning all data that it has been confronted with, which is not the case for BAO. Lastly, our refinement of detecting critical queries shows that the model improves. Deploying this model refinement naturally implies labeling and retraining phases. However, these phases can be handled asynchronously.

## 5 Summary and Outlook

In this paper, we showed that proper hinting of SQL queries in PostgreSQL can have a positive impact on the overall query execution time. For that, we started by describing PostgreSQL and BAO [Ma22], the state-of-the-art approaches of predicting hint sets, and elaborated on its shortcomings, which we tackled in our novel approach called *POSGB*. *POSGB* is a learning-based approach based on simple gradient boosting, where we learn one model per query context for fine-grained predictions rather than a single global context-agnostic model as done in BAO. In our evaluation, we demonstrated that hinting with *POSGB* produces better QEPs than BAO using the Stack benchmark [Ma22]. In particular, *POSGB* is consistently better than BAO resulting in much lower workload execution times over all training splits – closer to the optimal solution found throughout labeling. Nevertheless, we have not yet reached the optimal solution and rely on hintable, sub-optimally performing query compilers, which offers enough potential for further work in this area.

## References

- [Ch98] Chaudhuri, S.: An Overview of Query Optimization in Relational Systems. In (Mendelzon, A. O.; Paredaens, J., eds.): Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA. ACM Press, pp. 34–43, 1998, URL: <https://doi.org/10.1145/275487.275492>.

- [He22] Hertzschuch, A.; Hartmann, C.; Habich, D.; Lehner, W.: Turbo-Charging SPJ Query Plans with Learned Physical Join Operator Selections. *Proc. VLDB Endow.* 15/11, pp. 2706–2718, 2022, URL: <https://www.vldb.org/pvldb/vol15/p2706-hertzschuch.pdf>.
- [Ki19] Kipf, A.; Kipf, T.; Radke, B.; Leis, V.; Boncz, P. A.; Kemper, A.: Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In: 9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings. [www.cidrdb.org](http://cidrdb.org), 2019, URL: <http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf>.
- [Le15] Leis, V.; Gubichev, A.; Mirchev, A.; Boncz, P. A.; Kemper, A.; Neumann, T.: How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9/3, pp. 204–215, 2015, URL: <http://www.vldb.org/pvldb/vol9/p204-leis.pdf>.
- [Ma22] Marcus, R.; Negi, P.; Mao, H.; Tatbul, N.; Alizadeh, M.; Kraska, T.: Bao: Making Learned Query Optimization Practical. *SIGMOD Rec.* 51/1, pp. 6–13, 2022, URL: <https://doi.org/10.1145/3542700.3542703>.
- [Po] PostgreSQL: The World’s Most Advanced Open Source Relational Database, URL: <https://www.postgresql.org>.
- [Wo19] Woltmann, L.; Hartmann, C.; Thiele, M.; Habich, D.; Lehner, W.: Cardinality estimation with local deep learning models. In (Bordawekar, R.; Shmueli, O., eds.): *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019*, Amsterdam, The Netherlands, July 5, 2019. *ACM*, 5:1–5:8, 2019, URL: <https://doi.org/10.1145/3329859.3329875>.



# WEBTENSOR: Towards high-performance raster data analysis in the browser

Lucas Fabian Naumann<sup>12</sup>

**Abstract:** We present `WEBTENSOR`, a chunked tensor implementation for WebAssembly (`WASM`) compiled from a self-written C++ library and designed to efficiently analyze raster data directly in the browser. `WEBTENSOR` allows loading (chunked) data from various backends, manipulating it by aggregations and forwarding computed results in a zero-copy manner to `JAVASCRIPT` so that they can be further processed or visualized. We demonstrate the performance of `WEBTENSOR` by benchmarking data access and aggregation operations and compare it against a `JAVASCRIPT` version compiled from the same C++ code.

**Keywords:** WebAssembly; Raster Data; Tensor Processing; Visual Analytics

## 1 Introduction

With climate change research becoming increasingly important in the last years, so are raster datasets used in it, for example, from the Copernicus project<sup>3</sup> or the MOSAiC expedition<sup>4</sup>. Analysis of raster data is often done using visual analytics, a method where domain experts analyze the data with interactive visualization and exploration tools [Cu19]. Easy and platform-independent access to such tools could be realized by a browser application that is able to load the desired datasets and process them. However, such applications were not feasible in the past, as processing the data in the browser with `JAVASCRIPT` would be too inefficient due to performance limitations of the language, and doing the processing on the server side instead would introduce a too large overhead regarding requesting and receiving data [LH14]. This infeasibility changed when WebAssembly (`WASM`), a binary instruction format for a virtual machine, was launched in 2017 [Ha17]. With being supported by most browser engines and achieving a performance comparable to those of languages like C++ [Ja19], it is suited for high-performance data analysis in the browser.

So far, only a few data processing tools utilizing `WASM` have been proposed, like the embeddable SQL database DuckDB-Wasm [Ko22] or a `WASM` backend for TensorFlow.js<sup>5</sup>. None of those tools is suitable for analyzing raster datasets. Existing tensor implementations for browsers like TensorFlow.js focus on machine learning and thus lack features needed for analysis tasks. For example, tensors should be chunked to perform aggregations needed for the analysis efficiently and, as the data typically originates from a multitude of different sensors, data of varying backends, types and layouts should be processable in a single tensor.

---

<sup>1</sup> Technische Universität Dresden, lucas\_fabian.naumann@mailbox.tu-dresden.de

<sup>2</sup> German Aerospace Center, Institute of Data Science

<sup>3</sup> <https://copernicus.eu>

<sup>4</sup> <https://mosaic-expedition.org>

<sup>5</sup> <https://blog.tensorflow.org/2020/03/introducing-webassembly-backend-for-tensorflow-js.html>

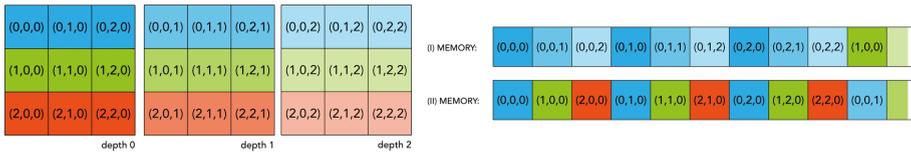


Fig. 1: Memory layouts for a three-dimensional tensor: (I) row-major order, (II) column-major order (row-/column-major order indicate data along the last/first dimension to be contiguous).

In this paper, we present `WEB_TENSOR`, a chunked tensor implementation for `WASM` designed for raster data analysis and intended to serve as a backend for `JAVASCRIPT` programs. Furthermore, we evaluate the performance of `WEB_TENSOR` on various data access and aggregation operations and compare the results against an equivalent tensor implementation in `JAVASCRIPT`.

## 2 Background

**Tensor Data Processing** Multidimensional data is often represented as a datacube, which, in its most basic form, is a tensor. In order to store multidimensional data, it is mapped onto the one-dimensional index space of storage devices (cf. Figure 1). The resulting memory layout has a significant impact on I/O performance. Due to the performance implications of data locality, multidimensional data is commonly chunked to reduce latency when the data access pattern might change over time<sup>6</sup>. Analyzing such data requires two kinds of queries: data accesses and aggregations. Data is accessed either at a single index of the tensor or along index ranges per dimension. The latter one is commonly called dicing or, if the index is static for one dimension, slicing. Aggregation reduces data along selected dimensions by applying a numeric operation. Consider, for example, a four-dimensional tensor with three spatial and one temporal dimension. Aggregating over the temporal dimension by taking a minimum, results in a three-dimensional tensor containing the minimum value over time at all spatial locations.

**WebAssembly & Compilation Toolchain** In the past, `JAVASCRIPT` has been the only programming language natively supported in browsers. Consequently, developing applications for the web required its usage, restricting the feasibility of computationally intensive applications because of its limited performance [Ha17]. In order to overcome this issue, `WASM`, a binary instruction format for a virtual machine usable in browsers, was introduced as a compilation target for high-level languages like `C++` [Ha17]. Jangda et al. showed that `WASM` is not only faster than `JAVASCRIPT` but even comparable to code executed natively on `x86` [Ja19]. Currently, the `WASM` heap is limited to 4 GiB in size as a 32-bit addressing space model is used [Ha17]. Furthermore, the interaction between `JAVASCRIPT` and `WASM` is one-sided since the `WASM` heap can be accessed by `JAVASCRIPT` but accessing `JAVASCRIPT` memory by `WASM` is not possible.

<sup>6</sup> [https://www.unidata.ucar.edu/blogs/developer/en//entry/chunking\\_data\\_why\\_it\\_matters](https://www.unidata.ucar.edu/blogs/developer/en//entry/chunking_data_why_it_matters)

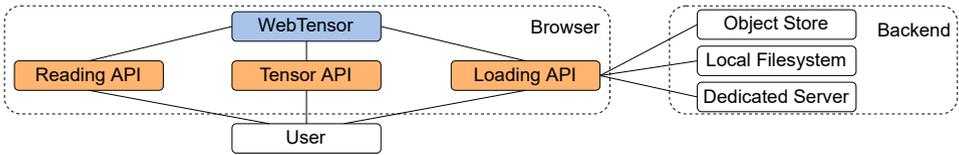


Fig. 2: Architecture of WEB TENSOR (orange: JAVASCRIPT components, blue: WASM components).

There are two major toolchains for compiling C++ code to WASM: Cheerp<sup>7</sup> (commercial) and Emscripten<sup>8</sup> (non-commercial, open source), with Emscripten achieving a better performance as reported by Yan et al. [Ya21]. Once installed, the Emscripten compiler frontend `em++` can be used as a drop-in replacement for regular C++ compilers. For the compilation process, Emscripten uses Clang and LLVM. Additionally to compiling to WASM, Emscripten allows to compile C++ to JAVASCRIPT.

### 3 Raster Data Analysis in the Browser

Figure 2 depicts an overview of WEB TENSOR and subsequent components (JAVASCRIPT components in orange, WASM components in blue). We compiled WEB TENSOR from a self-written C++ library to WASM using Emscripten. JAVASCRIPT programs can interact with it by utilizing three APIs, which were initially written in C++ and then compiled to WASM and bound to JAVASCRIPT methods using Embind<sup>9</sup>. These APIs enable WEB TENSOR to be used straightforwardly in JAVASCRIPT programs and allow arbitrary post-processing or visualization of tensor data, making it a flexible tool for various applications.

#### 3.1 WEB TENSOR

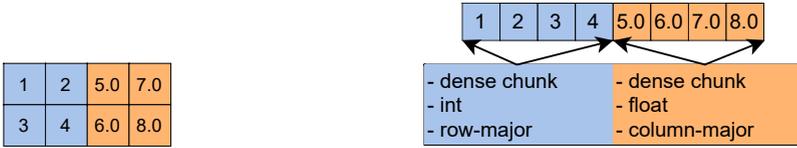
**Memory Layout** To maximize spatial data access locality, WEB TENSOR stores raw data in a binary buffer with data of fixed-sized rectangular chunks lying contiguously in it. For accessing the values corresponding to the bytes stored in this buffer, the tensor stores chunk objects, each one having pointers to one of those contiguous blocks and additional metadata, e.g., the chunk type, the value type of the data and the internal memory layout (cf. Figure 3). This decoupling of raw data and metadata has additionally the advantage that different chunk types (e.g., dense and sparse chunks), varying data types (e.g., float and int) and arbitrary internal memory layouts (e.g., row- and column-major) can be freely combined in a single tensor.

**Data Access** As the user should not have to care about the internal memory layout of WEB TENSOR, accessing its data is done by specifying indices regarding its dimensions, which are then transformed into offsets within the binary buffer. Such a transformation requires using the chunks as only these contain necessary metadata, like the type of the stored values. Since all chunks have a fixed shape, the one containing a user-provided index can easily be obtained and, with that, also an offset to the first element in it. In a second step, the metadata of the chunk can be used to determine the offset from its beginning.

<sup>7</sup> <https://leaningtech.com/cheerp>

<sup>8</sup> <https://emscripten.org>

<sup>9</sup> [https://emscripten.org/docs/porting/connecting\\_cpp\\_and\\_javascript/embind.html](https://emscripten.org/docs/porting/connecting_cpp_and_javascript/embind.html)



(a) Two-dimensional data.

(b) Buffer-Chunk structure of WEB\_TENSOR.

Fig. 3: Chunked data representation in WEB\_TENSOR (orange and blue color mark different chunks).

**Features** WEB\_TENSOR provides methods for accessing data at single indices as well as for dicing and slicing operations. Slicing and dicing operations thereby only return a view to existing data, which offers the same functionalities as a tensor regarding data access and aggregations and can be materialized to a new independent tensor at a later stage. Regarding aggregates, the computation of basic statistics, i.e., minimum, maximum, mean and standard deviation, on arbitrary dimensions is supported. Furthermore, it is possible to rechunk a tensor, thereby changing the in-memory order of its data, and thus favor access patterns and aggregations over specific dimensions.

### 3.2 APIs

**Loading** The Loading API enables JAVASCRIPT programs to load data from various backends to WEB\_TENSOR. For this, parts of Apache Arrow<sup>10</sup> (compiled to WASM) are used to process data in the Arrow IPC and Parquet format. Since WASM programs cannot access JAVASCRIPT memory, data should be loaded directly from the backend onto the WASM heap to avoid unnecessary copies. Currently, WEB\_TENSOR offers loading data in this way from a dedicated server using a WebSocket connection and the IPC format for serialization. Loading data in the Parquet format from object stores or the local file system is also possible, but at the moment, only by loading the data first in JAVASCRIPT and then copying it to WASM using the API, thus having an increased overhead.

**Tensor** This API binds the slicing, dicing and materializing, as well as the aggregate functions of WEB\_TENSOR to JAVASCRIPT methods using Emscriptens Embind. Hence, it enables the manipulation of tensor data and the construction of new views and tensors from the JAVASCRIPT side.

**Reading** Using this API, WEB\_TENSOR data can be accessed from JAVASCRIPT programs, where further processing or visualization (e.g., with the library D3<sup>11</sup>) is possible. Accessing the data is done by first obtaining its begin and end addresses in the corresponding chunks, as well as information about the chunk types, data types and memory layouts. Then, zero-copy, typed views of the data on the WASM heap are created with this information and returned to the JAVASCRIPT side through the API.

<sup>10</sup> <https://arrow.apache.org>

<sup>11</sup> <https://d3js.org/>

| Dice Shape        | WASM [ms] | JS [ms] |
|-------------------|-----------|---------|
| [400, 20, 20, 20] | 11.01     | 27.41   |
| [20, 20, 20, 20]  | 1.34      | 2.78    |
| [20, 30, 72, 72]  | 1.00      | 1.55    |
| [20, 30, 20, 72]  | 0.94      | 1.49    |

Tab. 1: Runtimes for dicing varying shapes.

| Aggregated Dimensions | WASM [ms] | JS [ms] |
|-----------------------|-----------|---------|
| time, alt, lat, lon   | 444.4     | 2439.0  |
| time, alt, lat        | 1136.4    | 2439.0  |
| time, alt             | 980.4     | 2777.8  |
| time                  | 574.7     | 3030.3  |

Tab. 2: Runtimes for aggregating the minimum over varying dimensions.

## 4 Experimental Evaluation

**Setup & Methodology** We compare WEB TENSOR against a JAVASCRIPT baseline implementation, compiled from the same C++ code using Emscripten, for various data access and aggregation operations. As baseline to compare against, we use a compiler-generated JAVASCRIPT implementation as it has been shown to consistently outperform equivalent manual implementations [Ya21]. We executed all benchmarks with benchmark.js<sup>12</sup> on a machine with an Intel i5-6440HQ CPU @2.6 GHz and 32 GiB RAM using a Firefox browser (version 107.0). As noted before by Yan et al., the specified optimization options for the compilation of the WASM and JAVASCRIPT code sometimes show unexpected behaviour, e.g., building with -O1 results in more efficient code than with -O3 [Ya21]. We used, in all cases, the -Os optimization flag, as it led to consistently good performance results.

**Dataset** For our evaluation, we use a space weather dataset provided by the German Aerospace Center. This dataset has four dimensions: time, altitude (alt), longitude (lon) and latitude (lat), along with multiple variables. The data is organized in row-major order, i.e., the values for varying latitudes lie contiguously in memory. With about 13 GB, the dataset is too large to be processed at once in WASM with its 32-bit addressing space model. It is planned to support processing datasets larger than 4 GiB by implementing a lazy loading strategy for dataset chunks and replacing the least recently used one when no further memory is available. However, this has not been implemented yet. To still show the performance of WEB TENSOR, we restrict ourselves to a data size of 250 MB by only regarding a [400,30,72,72]-shaped region of the dataset and one of its 32-bit floating point variables. The original dataset is not chunked, but for the benchmarks, we rechunk the data into 12 chunks of shape [100,30,40,40] (all dense chunks, with row-major order and floats as value type) having a size of approximately 21 MB.

**Benchmarks** First, we measured the access times for single indices. For this, we generated 100 random indices, measured their mean access times individually, and afterwards, took the mean over the 100 values received in this way. The resulting mean access time amounts to  $1.35 \cdot 10^{-2}$  ms for WASM and  $1.63 \cdot 10^{-2}$  ms for JAVASCRIPT, with a standard error of 0.1% for both.

<sup>12</sup> <https://benchmarkjs.com>

After the access of single values, we evaluated the performance of dicing operations. We did this by specifying four fixed dice shapes and, in a similar fashion as before, randomly generated 100 concrete dices (with varying start and end points) for each of those shapes, measured their mean runtimes individually and then computed the mean of the resulting 100 values. The received runtimes are shown in Table 1, the standard error is omitted in the table as it amounted to less than 0.16% for all shapes and is thus neglectable.

Lastly, we measured the execution times of aggregation operations. Table 2 shows the execution times for aggregating the minimum over the tensor dimensions specified by the “Aggregated Dimensions” column of the table. Again, the standard error is with at most 1.6% neglectable and not shown. The results for computing the maximum, mean and standard deviation are similar, hence we omit them here due to lack of space.

**Discussion** The WASM version of WEBTENSOR always achieves better results than its JAVASCRIPT counterpart. For point data access, WASM outperforms JAVASCRIPT by 21%, and for dicing and aggregating, it is, on average, faster by 92% and 294%, respectively. The performances differ thereby not by a constant factor but vary. Furthermore, while the results for the WASM and JAVASCRIPT columns in Table 1 are expected due to data locality and the number of operations to be performed, this is only the case for the WASM values in Table 2 and not for the ones of JAVASCRIPT. As the number of aggregated dimensions in Table 2 decreases from top to bottom, so does the number of operations needed to aggregate over them. Thus, the runtimes are expected to decrease too, besides when aggregating over all dimensions, as in this case, the data to be considered lies contiguously in memory, allowing optimizations. This expected behaviour can be observed for WASM. Regarding JAVASCRIPT, however, equal runtimes were obtained when aggregating over all four dimensions as when only considering time, altitude and latitude. For fewer dimensions, the runtimes increased even further. Currently, no satisfying explanation for this discrepancy could be found, but as the same observation was made in multiple repetitions of the benchmarks, it should be subjected to further studies in the future.

## 5 Summary & Next Steps

We presented WEBTENSOR, a chunked tensor implementation for WASM capable of efficient raster data analysis in the browser. Our initial experimental results indicate that a WASM-based tensor implementation can significantly outperform comparable JAVASCRIPT implementations on raster data access and aggregation operations. As a next step, we plan to extend WEBTENSOR such that larger datasets and more complex data analysis tasks become possible. More specifically, we plan to implement lazy loading of chunks from the backend such that complete datasets can be analyzed, as well as loading data from other backends without making unnecessary copies. Additionally, we intend to implement more aggregation operations (e.g., resampling, computing histograms) and a data cube layer on top of WEBTENSOR to provide more metadata information (e.g., physical coordinates of tensor indices).

## Bibliography

- [Cu19] Cui, Wenqiang: Visual Analytics: A Comprehensive Overview. *IEEE Access*, 7, 2019.
- [Ha17] Haas, Andreas; Rossberg, Andreas; Schuff, Derek L.; Titzer, Ben L.; Holman, Michael; Gohman, Dan; Wagner, Luke; Zakai, Alon; Bastien, JF: Bringing the Web up to Speed with WebAssembly. *SIGPLAN Not.*, 52(6), 2017.
- [Ja19] Jangda, Abhinav; Powers, Bobby; Guha, Arjun; Berger, Emery D.: Mind the Gap: Analyzing the Performance of WebAssembly vs. Native Code. *CoRR*, abs/1901.09056, 2019.
- [Ko22] Kohn, André; Moritz, Dominik; Raasveldt, Mark; Mühleisen, Hannes; Neumann, Thomas: DuckDB-Wasm: Fast Analytical Processing for the Web. *Proc. VLDB Endow.*, 15(12), 2022.
- [LH14] Liu, Zhicheng; Heer, Jeffrey: The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20(12), 2014.
- [Ya21] Yan, Yutian; Tu, Tengfei; Zhao, Lijian; Zhou, Yuchen; Wang, Weihang: Understanding the Performance of Webassembly Applications. In: *Proceedings of the IMC'21*. 2021.



# Which Rules Entail this Fact? An Efficient Approach Using RDBMSs

Tim Gutberlet<sup>1</sup>, Janik Sauerbier<sup>2</sup>

**Abstract:** Knowledge graphs (KGs) are used to store information about relationships between real-world entities in various fields. Learned rules over KGs describe patterns of KGs and allow for knowledge inference. In this paper, we focus on the problem of identifying all rules that entail a certain target fact given a KG and a set of previously learned rules. This can enable link prediction as well as help explain connections between rules and (potential) facts. Solving this problem time-efficiently for large rulesets and KGs is a challenge. To tackle this challenge, we propose an approach relying solely on RDBMSs including indexing, filtering and pre-computing methods. Our experiments demonstrate the efficiency of our approach and the effect of various optimizations on different datasets like YAGO3-10, WN18RR and FB15k-237 using rules learned by the bottom up rule learner AnyBURL.

**Keywords:** Knowledge graphs; Relational databases; Link prediction; Explainability.

## 1 Introduction

Many practically relevant large KGs are incomplete. Therefore, the prediction of missing or additional information, also known as link prediction, is highly relevant in contexts such as biomedicine [Br20] or social networks [Wa18]. Rules describing patterns of KGs, which are learned by rule learning tools, can enable link prediction. The rule  $(X, \text{citizenOf}, \text{germany}) \leftarrow (X, \text{bornIn}, \text{mannheim})$  describes for example the implication that someone ( $X$ ) born in Mannheim is a citizen of Germany. To determine the confidence of a potential new fact (link), one must know from which rules it can be derived. The confidence of a fact refers to the probability that a particular fact within the graph is correct. In our example: If we know that  $X$  was born in Mannheim it can help us determine the confidence of the fact that  $X$  is a citizen of Germany. Knowing the confidence is needed for rule-based link prediction models [Me19]. Moreover, the confidence is used for building ensembles with link prediction embedding models [Me18] and for improving link prediction embedding models [Gu18]. Therefore, it is important to identify all previously learned rules of a KG entailing certain target facts. Moreover, it is generally helpful to be able to explain the connections between rules and (potential) facts. To illustrate the performance of traditional RDBMSs in solving the problem in case of large rulesets and KGs, we create and analyze an efficient approach to this problem using PostgreSQL. As further discussed in the preliminaries, we are interested in the non-recursive application of rules and in finding all

---

<sup>1</sup> University of Mannheim, tim.gutberlet@students.uni-mannheim.de

<sup>2</sup> University of Mannheim, janik.sauerbier@students.uni-mannheim.de

applicable rules for a given target fact. Therefore, our approach is limited to binary relations and conjunctive queries, in contrast to employing deductive databases. Our approach uses indexing, filtering and pre-computing methods tailored to the natural structure of a KG and its respective rules. We ran several experiments testing our approach on different KGs (YAGO3-10 [MBS14], WN18RR [De18] and FB15k-237 [TC15]). Additionally, we show how different database setups impact the performance for different rule lengths. For the rule learning, we use AnyBURL, a fast bottom up rule learner for KGs [Me19].

## 2 Preliminaries

A KG is a set of (subject, relation, object)-triples also called facts. There is a set of entities present as subjects and objects, as well as a set of relations in a KG. Here is an example KG with the entities *peter*, *anna* and *germany* and the relations *marriedTo* and *bornIn*.

$$\mathbf{KG} = \{(anna, marriedTo, peter), (peter, bornIn, germany)\}$$

For our purposes, we are concerned about first-order logic Horn rules, which describe patterns of KGs. Here are two example rules. We capitalize the variables and lowercase the constants representing entities.

$$(X, citizenOf, germany) \leftarrow (X, bornIn, mannheim) \quad (1)$$

$$(X, livesIn, germany) \leftarrow (X, marriedTo, A_1), (A_1, bornIn, germany) \quad (2)$$

Each rule has a head (e.g.,  $(X, citizenOf, germany)$ ) consisting of one atom and a body (e.g.,  $(X, bornIn, mannheim)$ ) consisting of one or more atoms. A grounding of a rule assigns values to all variables of the rule, resulting in a ground rule. A true body grounding refers to a grounding of the body of a rule for which all (ground) atoms appear in the KG. The problem we are solving is identifying all rules that entail certain target facts based on the KG and a previously learned set of rules. This means we are interested in whether there exists a true body grounding and the head atom can be unified with the target fact. To illustrate the problem, think of the following target fact.

$$\mathbf{Target\ fact} = (anna, livesIn, germany)$$

Rule (1) does not entail this fact because its head atom cannot be unified with the target fact. The head of rule (2) can be unified with the target fact by assigning  $X = anna$ . Therefore, it would entail the target fact if  $\exists A_1((anna, marriedTo, A_1) \wedge (A_1, bornIn, germany))$ . The KG contains the facts  $(anna, marriedTo, peter)$  and  $(peter, bornIn, germany)$ . Together with the target fact, this results in a true body grounding for rule (2) with  $X = anna$  and  $A_1 = peter$ . Therefore, rule (2) is part of the solution.

### 3 Proposed Approach

The given KG is stored in one table of the relational database (*kg\_table* with columns *sub* for subjects, *rel* for relations and *obj* for objects). The basic idea behind our approach is creating SQL queries which check whether a certain rule entails a certain target fact. Those individual queries are then combined using *UNION ALL* operations over all rules, where the head can unify with the target fact. To improve this basic idea, we employ several optimizations listed below. Every rule has a specific rule ID which is returned if the rule entails the target fact. This would be the query for rule (2) and the target fact given in the preliminaries:

```
SELECT rule_id_2 FROM kg_table t0, kg_table t1 WHERE t0.sub = anna AND t0.rel =
marriedTo AND t0.obj = t1.sub AND t1.rel = bornIn AND t1.obj = germany LIMIT 1;
```

#### 3.1 Database Structure

Alternative to having one big table for all facts, our approach uses one table for each relation in the KG, with two columns for the subjects and the objects.

To speed up the search and enable direct access to the table contents through B-trees, we employ unique clustered indexes. We duplicate the knowledge graph tables and use once (subject, object) as key and once (object, subject) as key for the indexes. Within the generated SQL statements, the tables with the order (subject, object) are used in case of a fixed subject. The tables with the order (object, subject) are used in the case of a fixed object or no fixed subject and object. This ensures that the indexes are used efficiently.

#### 3.2 Advanced Rule Pre-filtering

Firstly, we only consider rules that can unify with the target fact for the query. This can be done by naively testing each rule. However, it can also be done more efficiently. The easiest way is storing each rule under its head as key in a hash map. The head stays unchanged, but we use “X” and “Y” as variable descriptors.

Let  $(s, r, o)$  be a target triple with the entities  $s$  and  $o$  and the relation  $r$ . Then we only use the rules stored under the keys  $(X, r, Y)$ ,  $(s, r, X)$ ,  $(X, r, o)$  or  $(s, r, o)$  for the query of the target triple. Additionally, when it is true that  $s = o$ , we also use the key  $(X, r, X)$ . Instead of looping through  $n$  rules naively in  $O(n)$ , this allows for pre-filtering of the rules in  $O(1)$ .

#### 3.3 Pre-computing of Expensive Rules

As we will illustrate in our experiments (4.3), certain rules result in way more cost in terms of execution time than others. To address that, we pre-compute a portion of those rules. To

identify the most expensive rules, we gather an independent set of  $n$  target facts from the KG and run their queries with the *EXPLAIN ANALYZE* command to get the execution time for the individual rule sub-queries. Afterward, we rank all rules which appear in the results by their average execution time multiplied with their number of appearances in the results.

The top  $x\%$  of rules are then pre-computed, by calculating all potential assignments for variables in the rule head that form a grounding together with a set of facts from the KG. We then store these combinations in a table for the rule. If the subject or the object is a variable and the other one a constant, then the table only contains the column for the variable. For the implementation, we use indexed materialized views to *SELECT* from the KG with all the conditions we already know from the rule. Consequently, the *CREATE* statement for pre-computing rule (2) and the query for the target fact from the preliminaries would be:

```
CREATE MATERIALIZED VIEW view_rule_2 SELECT t0.sub AS sub FROM kg_table t0, kg_table t1 WHERE t0.rel = marriedTo AND t0.obj = t1.sub AND t1.rel = bornIn AND t1.obj = germany;
```

```
SELECT rule_id_2 FROM view_rule_2 WHERE sub = anna LIMIT 1;
```

We exclude rules with one body atom, as pre-computing does not improve their performance. Beyond that, we limit the pre-computation to rules with two body atoms for illustration purposes, as rules with more body atoms incur a pre-computation time two orders of magnitude higher than rules with two body atoms.

## 4 Experiments

The goal of our experiments is to benchmark the performance of our approach on different datasets and rulesets, as well as to measure the effects of different optimizations. Furthermore, we analyze the execution time for individual rules.

For the implementation of our approach, we used the open-source object-relational database system PostgreSQL. Additionally, we used Java with the PostgreSQL JDBC driver. The source code and datasets, we used for the experiments, can be found at <https://github.com/timgutberlet/Which-Rules-Entail-This-Fact>. We conducted all experiments on a Fujitsu Esprimo P957 (construction year 2017) with 32 GB RAM, 512 GB SSD, an Intel i7-7700 @ 3.6 GHz CPU and Ubuntu 22.04.1 LTS as an operating system. The rule learning with AnyBURL was mostly done with the standard configuration of AnyBURL-22 available at <https://web.informatik.uni-mannheim.de/AnyBURL/>. We only extended the limit of body atoms from one to three for acyclic rules to match the cyclic rules, as we do not intend to discriminate between them. This means the rulesets only include rules with up to three body atoms. The relations in the rule bodies are always extensional (defined by facts), never intensional (defined by other rules).

|           | AnyBURL - 10s | AnyBURL - 50s | AnyBURL - 100s |
|-----------|---------------|---------------|----------------|
| YAGO3-10  | 4.2           | 10.4          | 14.3           |
| WN18RR    | 34.8          | 65.0          | 77.3           |
| FB15k-237 | 5.4           | 26.2          | 42.9           |

Tab. 1: Performance results (avg. execution time per target fact in ms)

|           | #entities | #relations | #facts of KG | 10s | 50s  | 100s |
|-----------|-----------|------------|--------------|-----|------|------|
| YAGO3-10  | 123,182   | 37         | 1,079,040    | 27k | 107k | 165k |
| WN18RR    | 40,943    | 11         | 86,835       | 6k  | 20k  | 30k  |
| FB15k-237 | 14,505    | 237        | 272,115      | 33k | 135k | 252k |

Tab. 2: Dataset properties &amp; #rules per ruleset learned by AnyBURL

| Experiment                                | Avg. execution time (in ms) |
|---|-----------------------------|
| All optimizations enabled                 | 10.4                        |
| Advanced rule pre-filtering disabled      | 55.0                        |
| Tables for each relation disabled         | 132.9                       |
| Indexing disabled                         | 1720.2                      |
| Pre-computing of expensive rules disabled | 14.1                        |

Tab. 3: Ablation study of optimizations using YAGO3-10 &amp; 107k rules

## 4.1 Overall Performance

As illustrated in Tab. 1, our approach works for different datasets and rulesets learned by AnyBURL. For every dataset, we learned rules for 10s, 50s, and 100s (Tab. 2). We used the training sets as the KGs and the test sets as the target triples (3k-20k triples). The validation sets (3k-20k triples) were used as target triples to create the rule rankings to pre-compute the top 1% most expensive rules with two body atoms.

## 4.2 Ablation Study - Optimizations

For the ablation study in Tab. 3, we used YAGO3-10 as the benchmark dataset and the AnyBURL 50s ruleset (107k rules). We didn't analyze versions with multiple optimizations disabled due to very high execution times. The "tables for each relation" optimization described in 3.1 specifically reduces the execution time for long rules. We tested this, by measuring the average execution time of different rule lengths in the version where the "table for each relation" optimization is enabled versus the version where the "table for each relation" optimization is disabled. In both runs, the rule pre-computation was disabled. In this experiment, the execution time for rules with one body atom was reduced by 12%, for

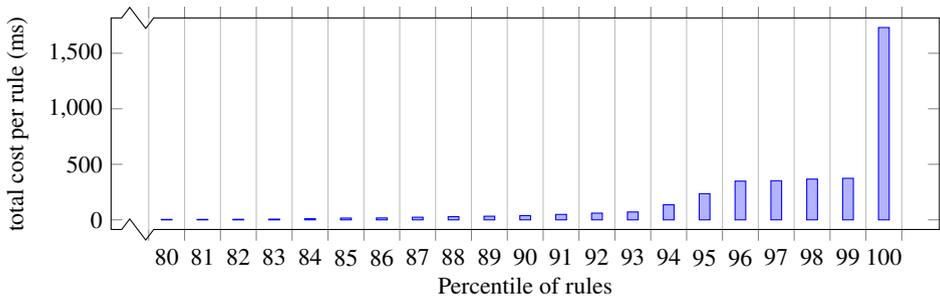


Fig. 1: Quantile performance analysis of rules using YAGO3-10 & 107k rules

two body atoms by 25% and for three body atoms by 74% when the “table for each relation” optimization is enabled. This might be caused by the increased use of the indexes during execution for longer rules.

### 4.3 Rule Quantile Performance Analysis

The quantile analysis in Fig. 1 is based on the total cost per rule after using the validation set as target triples, as described in 3.3. The total cost per rule is calculated by multiplying the number of appearances with the average execution time. Here, the number of appearances is a count for how often a rule appears as a result of the queries for all target triples. It indicates that only few rules incur a major share of the cost. The most expensive rules are predominantly rules with two variables in the rule head and two or three body atoms. Excluding those rules (978 rules) reduced the average execution time for YAGO3-10 & the AnyBURL 50s ruleset (107k rules) to 0.14 ms per query using all optimizations. In the given example (YAGO3-10 & 107k rules), we achieved an 98.8% execution time reduction for the pre-computed rules. The pre-computation took 15 minutes. This reduced our average execution time from 14.1 ms to 10.4 ms, as illustrated in Tab. 3.

## 5 Conclusions

We have designed an efficient approach for finding all rules that entail a certain target fact given a knowledge graph and a set of previously learned rules. Our experiments specifically demonstrate the effect of indexing, filtering and pre-computing methods. Potential next steps include a further analysis of our approach on various datasets, an empirical comparison of different database technologies (particularly triplestores and deductive DBMS), exploring the use of multithreading, investigating the use of multidimensional indexes and creating a dedicated solution only using the main memory.

**Acknowledgement.** This paper would not have been possible without the exceptional support of our supervisor, Prof. Dr. Rainer Gemulla.

## Bibliography

- [Br20] Breit, Anna; Ott, Simon; Agibetov, Asan; Samwald, Matthias: OpenBioLink: a benchmarking framework for large-scale biomedical link prediction. *Bioinformatics*, 36(13):4097–4098, 2020.
- [De18] Dettmers, Tim; Minervini, Pasquale; Stenetorp, Pontus; Riedel, Sebastian: Convolutional 2d knowledge graph embeddings. In: *Proceedings of the AAAI conference on artificial intelligence*. volume 32, 2018.
- [Gu18] Guo, Shu; Wang, Quan; Wang, Lihong; Wang, Bin; Guo, Li: Knowledge Graph Embedding With Iterative Guidance From Soft Rules. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.
- [MBS14] Mahdisoltani, Farzaneh; Biega, Joanna; Suchanek, Fabian: Yago3: A knowledge base from multilingual wikipedias. In: *7th biennial conference on innovative data systems research*. CIDR Conference, 2014.
- [Me18] Meilicke, Christian; Fink, Manuel; Wang, Yanjie; Ruffinelli, Daniel; Gemulla, Rainer; Stuckenschmidt, Heiner: Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion. In: *The Semantic Web – ISWC 2018*. pp. 3–20, 2018.
- [Me19] Meilicke, Christian; Chekol, Melisachew Wudage; Ruffinelli, Daniel; Stuckenschmidt, Heiner: Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI'19*. pp. 3137–3143, 2019.
- [TC15] Toutanova, Kristina; Chen, Danqi: Observed versus latent features for knowledge base and text inference. In: *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*. pp. 57–66, 2015.
- [Wa18] Wang, Zhouxia; Chen, Tianshui; Ren, Jimmy; Yu, Weihao; Cheng, Hui; Lin, Liang: Deep Reasoning with Knowledge Graph for Social Relationship Understanding. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence. IJCAI'18*, p. 1021–1028, 2018.



# Explainable Data Matching: Selecting Representative Pairs with Active Learning Pair-Selection Strategies

Lukas Laskowski, Florian Sold<sup>1</sup>

**Abstract:** In both research and enterprise, dirty data poses numerous challenges. Many data cleaning pipelines include a data deduplication step that detects and removes entries within a given dataset which refer to the same real-world entity. Throughout the development of such deduplication techniques, data scientists have to make sense of the large result sets that their matching solutions generate to quickly identify changes in behavior or to discover opportunities for improvements. We propose an approach that aims to select a small subset of pairs from the result set of a data matching solution which is representative of the matching solution's overall behavior. To evaluate our approach, we show that the performance of a matching solution trained on pairs selected according to our strategy outperforms a randomly selected subset of pairs.

**Keywords:** Entity Resolution; Data Matching; ExplainableDM; Pair Selection; Benchmark

## 1 Explainable Data Matching

Improving data matching systems is an iterative process: Insights on matching behavior derived from the set of output labels of the matching solution serve as the basis for improvements in the next iteration. To accelerate this optimization process, we have developed a data matching benchmark platform, called Frost [Gr22]. Frost combines existing benchmarks, established quality metrics, cost and effort measures for evaluating and comparing data matching solutions. Furthermore, the platform includes techniques which enable the systematic exploration of matching results. However, as real-world datasets can contain millions of records, it is unrealistic to examine all pairs within a result set. Consequently, there is the need to summarize data matching results such that only a representative subset of the most meaningful pairs remains. Based upon a matching result set of size  $m$  generated with a data matcher on a dataset of size  $n$ , we aim to select a subset of well-distinguishable pairs of size  $k$  with  $k \ll m$  that are representative of a matching solution's behavior.

To achieve this goal, we leverage instance selection strategies from the field of active learning, a semi-supervised learning method, where the initial seeded training dataset is very small or empty. Therefore, to sufficiently train the data matching classifier, more label

---

<sup>1</sup>Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam,  
{lukas.laskowski, florian.sold}@student.hpi.de

candidates are iteratively selected and, after a manual labeling process, added to the training dataset. Since manual labeling is costly, a variety of pair selection approaches exist that aim to achieve a target matching quality in as few iterations as possible. Our goal to select a subset of pairs which summarize the matching classifier as good as possible is similar to the pair-selection task in active learning. Nevertheless, we differ in how the pair-selection algorithm is applied since our use case works upon a fully labelled result set and therefore lacks the iterative procedure.

Within the field of active learning for data matching, Christen et al. [CCR20] recently proposed an iterative pair-selection strategy based upon their novel informativeness measure, which outperforms prior works. Hence, we base our non-iterative pair-selection approach upon the informativeness measure by Christen et al. With an informativeness score for each pair, we choose the representative subset as the set of pairs with the highest score.

In the following sections, we first introduce related work to explainable data matching (Section 2) and then introduce our proposed approach (Section 3). Next, in Section 4, we show the effectiveness of this strategy by experimentally analyzing its behavior. Finally, we conclude and discuss next steps (Section 5).

## 2 Related Work

Explaining data matching decisions is generally a challenging task. Especially with complex machine learning or deep learning models like DITTO [Li21], results are difficult to interpret but often superior to those achieved with approaches based upon simple classifiers or rule-based approaches. Without specific tools, it is close to impossible to understand how a certain black-box matcher assigns labels, as the underlying model can be very complex. Additionally, result sets are large and make it difficult to select pair instances for further analysis. While explanation techniques already exist in the machine learning community, Thirumuruganathan et al. found that these techniques do not suit the needs of data matching scientists and propose a variety of research opportunities [TOT19].

Baraldi et al. [Ba21] propose, with their Landmark explanation framework, a new framework for local pair-specific explanations specifically tailored towards data matching. Landmark introduces two main innovations: First, it generates per pair two explanations by fixing either record (called landmark) and perturbing the other record (called varying entity). Second, it produces an artificial entity by attribute-wise concatenating both entities. The MOJITO framework [Di19] analyzes the influence of individual attributes on the matching decision. Their results show that black-box entity matching models might rely on untrustworthy attributes. Hence, they conclude that quality metrics are not sufficient to quantify real-world performance of a data matching model, but rather emphasize the need for explanations.

Nevertheless, surprisingly little work has been done so far towards matching solution-agnostic filtering or selection of representative (or summarized) result subsets. Explanation frameworks like Landmark can then be executed upon pairs within a representative subset.

### 3 Informativeness-based Selection of Pairs

To explain the behavior of a matcher, we present an approach that selects pairs out of the result set labeled by a matching solution. As described in Section 1, we base our selection strategy upon the informativeness measure developed by Christen et al. [CCR20].

$$\text{informativeness}(u, R) = (1 - \alpha) * \text{entropy}(u, R) + \alpha * \text{uncertainty}(u, R)$$

The informativeness-score for  $u$  in respect to the result set  $R$  is the weighted average of the uncertainty and the entropy. The balancing factor is given as  $\alpha$  and set to  $\alpha = 0.5$ .

To each labeled pair within  $R$ , we assign a similarity vector  $u$ , which consists of the pair-wise similarity for each attribute of the two records  $r_1, r_2$  with  $(r_1, r_2) \in R$ . To calculate the informativeness, we use an additional similarity measure between a pair of pair vectors  $u, v$ , in our case cosine similarity, called  $\text{sim}(u, v)$  that we assume to already exist. For a pair  $u$ , we define the set  $R_S$  as all pairs from the result set that were assigned the same label as  $u$  and as  $R_O$  those assigned the opposite label. An environment  $S$  of supporting pairs for a pair  $u$  is bounded by the closest instance classified contrary to the target pair, and defined as:

$$S(u, R) = \{v \in R_S | \text{sim}(u, v) > \max\{\text{sim}(u, w) | w \in R_O\}\}$$

$$\text{uncertainty}(u, R) = \frac{1}{1 + |R \cap S(u, R)|}$$

Uncertainty quantifies how certain a label assignment is. In case only very few similar pairs reside within the environment  $S$  around a target pair, the label assigned to it is uncertain. Hence, its uncertainty score will be higher. Compared to a pair with plenty of confirmations, a pair with high uncertainty might be especially valuable and representative of the matching classifier's behavior.

$$\text{entropy}(u, R) = - \left[ \frac{\sum_{v \in R_{SE}} u * \text{sim}(u, v)}{|R| - 1} * \log\left(\frac{\sum_{v \in R_{SE}} u * \text{sim}(u, v)}{|R| - 1}\right) + \frac{\sum_{v \in R_{OE}} u * \text{sim}(u, v)}{|R|} * \log\left(\frac{\sum_{v \in R_{OE}} u * \text{sim}(u, v)}{|R|}\right) \right]$$

The entropy value represents how diverse the environment around a certain pair  $u$  is. It can only yield a high value in case both summands are high negative values. Again, this can only be true in case we have both very similar pairs of the same and the opposite class. Pairs that fulfill this requirement are considered to have high entropy, as they help to form the decision boundary and are therefore, again, representative of the classifier's behavior. Compared to the active learning setting, our approach does not work iteratively, and we do have already all pair labels available (as assigned by the matching solution). Therefore, we would need to consider all pairs for entropy calculation, which in return would lead to a very similar entropy value for all pairs. Hence, we restrict the pairs considered for entropy calculation to those pairs whose similarity to the target pair  $u$  is higher than the entropy environment limit  $e$ . We define the environment boundary  $e$  relative to the S-Environment

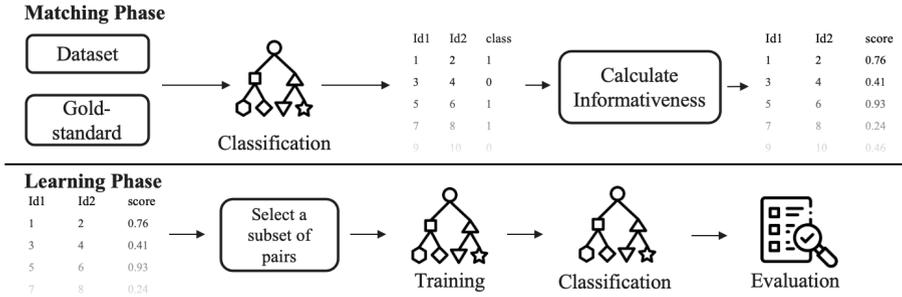


Fig. 1: **Evaluation Process.** This figure shows the two phases of our evaluation approach: “Matching Phase” and “Learning Phase”.

as  $e(u, R) = S(u, R) * l$  with  $l \in [0, 1]$  and set to  $l = 0.4$ . The set  $R_{SE}$  includes all pairs of the same class as  $u$  and a similarity larger than  $e(u, R)$  to  $u$ . Similarly, we define the set  $R_{OE}$  with pairs of opposite class. Since the entropy base metric requires a similarity value for all pairs of pairs, the proposed approach has a runtime complexity of  $O(n^4)$  with  $n$  as the amount of records in the base dataset.

Under the assumption that machine learning models benefit similarly from a representative pair subsets as humans do, we select the top  $k$  pairs by informativeness measure as the representative subset. Hence, this particular subset summarizes the matching behavior of the matching solution better than any other equally sized subset could. In case one chooses  $k$  small enough, the selected pairs now serve as a basis for a human analysis.

## 4 Evaluation

We evaluate our approach by training a matching solution only upon the ground truth labels of pairs within the representative subset, and then comparing its quality using F1-Score against two baselines. Figure 1 outlines the overall evaluation-process: We first produce results for informativeness-calculation by predicting labels on unseen testing data, which serve as the basis for the informativeness score (“Matching Phase”). During the subsequent “Learning Phase”, we select the  $k$  pairs with the highest informativeness score as our subset of pairs. These pairs and their respective ground truth label are then used to train the same (but untrained) classification model. Afterwards, we predict and evaluate against the full gold standard.

We perform experiments using the matching solution “Cyber-Punk” which is one of the winning concepts of the SIGMOD programming contest in 2021<sup>2</sup>. As the dataset, we used “SIGMOD AltoSight Z4”, which contains mainly textual data about SD cards. Besides our informativeness selection strategy, we compare up with two naïve baseline subsets:

<sup>2</sup> <https://dbgroup.ing.unimore.it/sigmod21contest/index.shtml>

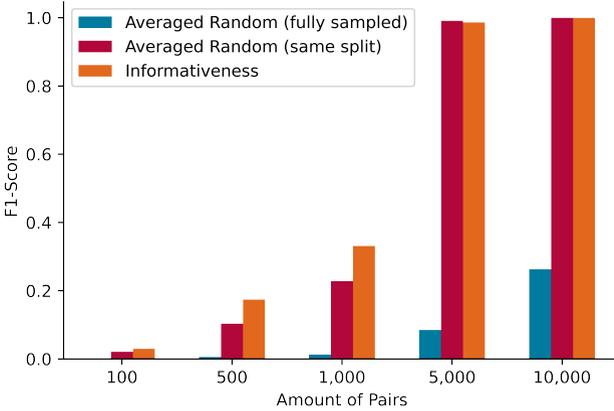


Fig. 2: **Retraining on Selected Pairs.** Comparing the results for the presented selection strategies on multiple subset sizes on dataset *Z4* using the “Cyber-Punk” matcher. We use subset sizes of  $k \in \{100, 500, 1000, 5000, 10000\}$ .

- **Averaged Random (fully sampled):** For this strategy, we sample randomly  $k$  different pairs out of the set of pairs. In practice, this resembles a human who scrolls through the entire result set with no further filters or selections available.
- **Averaged Random (same split):** In most datasets, the majority of pairs are easy to label as non-matches. Therefore, the previous strategy predominantly selects such pairs and only few (if any) duplicate pairs. In contrast, this strategy samples  $k$  pairs with the same proportion of duplicates and non-duplicates (according to the ground truth annotation) as can be found in the selected subset. Although we sample randomly, this subset implicitly comes with a substantial advantage as its proportions are based upon the informativeness-based subset with respect to the ground truth annotation.

As shown in Figure 2 the informativeness-based selection strategy indeed outperforms the completely randomly sampled selection strategy at any subset size. This observation is reasonable, as likely many trivial non-matches were sampled into the subset. Consequently, a smaller subset size  $k$  causes a larger difference between the two solutions: At a subset size of 100, the informativeness-based strategy ( $f_1 = 0.03$ ) has a 15x higher F1-Score compared to the randomly based selection strategy ( $f_1 = 0.002$ ). With more pairs in the subset, we see a similar effect. For instance, the informativeness-based strategy ( $f_1 = 0.986$ ) has a subset size of 5,000 an F1-Score which is 11.6x higher than the randomly based selection strategy ( $f_1 = 0.085$ ). Even in comparison to the random sampling of *same split*, our approach achieves the highest scores on small subsets and matches on subsets of size  $k \geq 5000$ .

Since our use-case is mainly targeted towards subsets of size 1000 or less that a human can

grasp or look through, and we do significantly outperform both sampling approaches in this area, our approach does indeed work as anticipated. This shows that a subset selected using the informativeness score includes pairs that bear more information compared to random sampling – and therefore offers insights into the behavior of a matching solution.

## 5 Conclusion and Outlook

We set out to select a representative subset of pairs out of a potentially very large result set. Our results do indeed indicate that the informativeness-based subset outperforms random selection by far. More importantly, these results now serve as a baseline for further improvements towards reducing the overall runtime or developing novel approaches.

Beyond our current results, we see further research opportunities in this area. For example, our existent evaluation can be underlined with additional test settings including multiple datasets as well as a diverse set of matching solutions from various domains. Furthermore, one could further extend this approach by indicating for each selected pair how many similar non-selected pairs exist in the result set. In case these pairs were accessible to a data scientist, he could better make sense of this particular pair’s properties.

**Acknowledgements.** This paper is the result of a seminar supervised by Felix Naumann and Luca Zecchini. We thank them for their valuable input and support during our project.

## References

- [Ba21] Baraldi, Andrea; Del Buono, Francesco; Paganelli, Matteo; Guerra, Francesco: Landmark Explanation: An Explainer for Entity Matching Models. In: Proceedings of the International Conference on Information and Knowledge Management. ACM, pp. 4680 – 4684, 2021.
- [CCR20] Christen, Victor; Christen, Peter; Rahm, Erhard: Informativeness-Based Active Learning for Entity Resolution. In: European Conference on Principles of Data Mining and Knowledge Discovery. Springer International Publishing, pp. 125–141, 2020.
- [Di19] Di Cicco, Vincenzo; Firmani, Donatella; Koudas, Nick; Merialdo, Paolo; Srivastava, Divesh: Interpreting Deep Learning Models for Entity Resolution: An Experience Report Using LIME. In: Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. ACM, 2019.
- [Gr22] Graf, Martin; Laskowski, Lukas; Papsdorf, Florian; Sold, Florian; Gremmelspacher, Roland; Naumann, Felix; Panse, Fabian: Frost: a platform for benchmarking and exploring data matching results. PVLDB, 15(12):3292–3305, 2022.
- [Li21] Li, Yuliang; Li, Jinfeng; Suhara, Yoshihiko; Doan, AnHai; Tan, Wang-Chiew: Deep Entity Matching with Pre-Trained Language Models. PVLDB, 14(01):50–60, 2021.
- [TOT19] Thirumuruganathan, Saravanan; Ouzzani, Mourad; Tang, Nan: Explaining Entity Resolution Predictions: Where are we and What needs to be done? In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics. ACM, pp. 1–6, 2019.

# Efficient handling of recursive relationships in ORM frameworks using Entity Framework Core as an example

Benjamin Uwe Killisch,<sup>1</sup> Florian Scheffler,<sup>2</sup> Thomas Kudraß<sup>3</sup>

**Abstract:** ORM frameworks are a popular method to bridge the differences between object-oriented programming and relational data management. At the same time, recursive relationships are present in many schemas to represent tree-like or net-like structures. This paper discusses how to efficiently build and execute queries for data with recursive relationships in ORM frameworks, mainly Entity Framework Core (EF Core). Five possible solutions are conceived and implemented, while making sure that they can be used like regular LINQ queries. Next, the solutions are tested with different SQL dialects. The results of these tests are then analyzed by a variety of test parameters. This analysis shows that queries with recursive common table expressions and queries using key loading are the most efficient. Queries with auxiliary property, vertical unrolling or horizontal unrolling, are either too slow or only usable under particular circumstances. The analysis also shows that the performance of the solutions is always dependent on the circumstances, especially the SQL dialect.

**Keywords:** Object-Relational Mapping; Recursive; Relationships; Queries

## 1 Introduction

Object-oriented programming and relational databases are currently the most popular paradigms for programming and persistent data storage. Since they are different, object-relational mapping (ORM) is needed to bridge their differences when using them together. One challenge when using an ORM framework is loading data with recursive relationships from the database. The motivation for this paper was the challenge of loading article attributes in a recursive relationship for a REST API developed by the e-commerce company Relaxdays GmbH. Performance is critical, because hundreds of articles have to be updated per day. The topic of recursive queries has already been covered by Szumowska et al. [Sz11] for the framework Hibernate. This paper will cover Entity Framework Core (EF Core) for C# [Mi21]. EF Core can automatically map classes and their relationships to a database, and convert the results of database queries into objects. Its most distinguishing feature is LINQ (Language-Integrated Query) [Mi22a]. Using LINQ, one can create database queries using method chains and type-safe lambda expressions based on the classes mapped to the database. There is no need to write SQL queries as strings, and therefore, EF Core can be used independently of the underlying database.

---

<sup>1</sup> HTWK Leipzig, Fakultät Informatik und Medien, benjamin\_uwe.killisch@stud.htwk-leipzig.de

<sup>2</sup> Relaxdays GmbH, florian.scheffler@relaxdays.de

<sup>3</sup> HTWK Leipzig, Fakultät Informatik und Medien, thomas.kudrass@htwk-leipzig.de

## 2 Solutions

There are multiple requirements for possible solutions. Solutions should be as fast as possible, while also being convenient to use. They should also be able to handle cyclical data and still terminate while returning the correct result. In the context of EF Core, the solutions should function regardless of the SQL dialect. Furthermore, they should accept two lambda expressions, one for the initial condition and one for the navigation property of the recursive relationship. Navigation properties represent relationships between two classes that are mapped to tables in the database via EF Core. The first class will have a navigation property that is a single reference or a list of the second class. An example of this is shown in Listing 1. The query will be created for the recursive relationship represented by the navigation property `employee.Subordinates`.

List. 1: Usage of the solution implemented in section 2.1.

```
context.Employees.RecursiveQuery(context, employee => employee.EmployeeId == 1, employee =>
    employee.Subordinates);
```

### 2.1 Recursive CTEs

Common Table Expressions (CTEs) were specified in the SQL:1999 standard and allow for queries to be named and reused again later, as described by Heuer et al. [HSS18]. CTEs can be used as recursive queries by adding the **RECURSIVE** keyword. A recursive CTE consist of the initial query, the recursive query, and the actual query. Only the recursive query can reference the CTE it is a part of. If we use this to join the CTE with the table we want to query, we can query the table recursively. The recursive CTE can then be referenced in the actual query. There are some problems with this solution, the main one being that the syntax of recursive CTEs is not standardized across the different SQL dialects [SQ22] [Or22] [Po22] [Mi22b]. Furthermore, not all SQL dialects can handle cyclical data properly. Out of the four dialects examined in this paper, Transact-SQL can not, but the other three can. The query will terminate with an error after reaching the maximum recursion depth of 1000. Furthermore, CTEs can not be used as subqueries in Transact-SQL. Recursive CTEs can be used in an ORM framework like any other query, although they are more complex. It gets more complicated when using a framework like EF Core. The following steps are executed to transform two lambda expressions into a recursive CTE:

1. The navigation property of the recursive relationship needs to be read from the first lambda expression. EF Core stores the mapping of the classes to the database schema, including the relationships between tables, so the columns of the relevant table and the corresponding primary keys and foreign keys can be loaded from there.
2. The second lambda expression, which contains the initial query, must be converted into a query string. For that purpose, EF Core provides the `ToQueryString()` method.

3. The resulting string must be slightly adjusted to account for parameterized queries. The values of query parameters must also be read from the second lambda expression.
4. Based on the type of the relationship (1:1, 1:n, m:n) and the schema data loaded in the first step, the relevant table names, column names and key comparisons are selected.
5. The final query string is built while considering the used SQL dialect.
6. The final query string and the parameter values are passed to `FromSQLRaw()`.

`FromSQLRaw()` returns an instance of `IQueryable` (the interface used to query data sources), which can be chained to further LINQ methods to extend the query. However, EF Core only allows this if the query string passed to `FromSQLRaw()` starts with `SELECT`. A CTE always starts with the keyword `WITH`, so the query string has to be surrounded with `SELECT * FROM (query)`. But, as mentioned above, CTEs may not be used as subqueries in Transact-SQL. So, if Transact-SQL is used, the query is immediately evaluated, and an `IN` subquery based on the returned objects is created. Not only does this worsen the performance, but it also violates lazy evaluation, which is usually expected from instances of `IQueryable` in C#. An alternative would be to keep track of the search path in an additional column, and to check that column in the `where` clause of the recursive query. This, however, has not been implemented for this paper.

## 2.2 Vertical Unrolling

Boniewicz et al. [BSW12] describe alternatives to CTEs for recursive queries. One of them is vertical unrolling, which works by combining multiple `LEFT JOINS`. The number of joins is equal to the maximum recursion depth of the table. In the best-case scenario, this depth is known. Otherwise one must make an educated guess with the risk of incomplete query results. The size of the query increases with the depth of the recursive structure, which can make this very impractical to implement when using query strings. In EF Core, however, this solution can be implemented easily with a loop and the `ThenInclude()` method. In EF Core, `Include()` and `ThenInclude()` are used to load related data and are translated to `JOINS`. The query method accepts the recursion depth as an additional parameter and calls `ThenInclude()` accordingly.

## 2.3 Horizontal Unrolling

In addition to vertical unrolling, Boniewicz et al. [BSW12] also describe horizontal unrolling. Horizontal unrolling creates a temporary table for each level of the recursive structure, creating each one based on the previous table. These temporary tables are then combined using `UNION`. Similarly to vertical unrolling, this approach also requires knowledge about the maximum recursion depth. It can be implemented using a loop, and the resulting

queries will be simpler than vertical unrolling. However, temporary tables are not convenient to use in EF Core. Among other things, the name and columns of a temporary table must be known at compile time, making them impractical since one needs a specific temporary table for each entity and recursion level. Because of this, a different approach is implemented in this paper. The queries for the temporary tables are all executed immediately. The resulting entities are then used to create the next query. In the end, a final IN subquery is created based on all the loaded entities. This approach creates a lot of database roundtrips, but makes the recursion-depth parameter obsolete. Instead, a new query is only executed if the last query returned at least one unknown entity.

## 2.4 Using an auxiliary property

In some cases, the table with the recursive relationship has a column for which all records of the same recursive structure share the same value. Such a column can be used to create a simple recursive query. First, all the values of the aforementioned column (the auxiliary property) are loaded for the records that match the initial query. Then, an IN subquery for the auxiliary property is executed based on the loaded values. This is exactly how this approach is implemented in EF Core. While this approach is simple, it has a few drawbacks:

1. Such a column must either exist already in the current schema or be added to it. If it is added, it also needs to be maintained on every insert or update operation.
2. The auxiliary column should have an index to improve the performance.
3. This approach always loads the entire recursive structure, even if only a part of it is required.
4. To use this approach for an m:n relationship, one would have to create an additional table that links every record in the original table to the roots of the structures it is a part of. Since this would make the query more complex, and such a table would be much harder to maintain than just a column, this approach is only implemented for 1:1 and 1:n relationships.

## 2.5 Key loading

The idea of key loading is to load the values of the primary and foreign keys of all records in the table, and then use this data to find the primary key values of the records in the recursive structures to load. For this, the primary keys of the records matching the initial query must also be loaded. Finally, the relevant records are loaded with an IN subquery. This approach uses the fact that object-oriented programming languages are turing complete, while SQL is not. A disadvantage of this approach is that a lot of data could be loaded unnecessarily if only a small amount of records of a big table is required. Usually, one can easily implement

this approach with three queries. In EF Core, it is a bit more complex, mainly because simple primary keys must be handled as well as composite primary keys. To do this, both kinds of keys are represented as value tuples. This has the advantage that C# compares value tuples by value, not by reference.

### 3 Performance comparison

Each one of the described approaches was tested in many different scenarios, varying in SQL dialect (MySQL, SQLite, PostgreSQL, Transact-SQL), relationship type (1:1, 1:n, m:n), recursion depth (3, 4), the amount of branches per layer (2, 5, always 1 for 1:1 relationships) and the amount of recursive structures in the database (5, 10, only one structure was loaded per query execution). Every table usually had one or two columns additionally to the key columns. Every approach was repeated and measured 150 times for each resulting scenario. For vertical unrolling, the correct recursion depth was passed to the query method. This is the ideal scenario, while in reality one might have to use a higher maximum recursion depth, to make sure to always load all the data. All tests were executed using an Intel® Core™ 5 i7-10510U Processor and 32 GB RAM. The databases for MySQL, Transact-SQL and PostgreSQL were hosted in docker containers, while the SQLite database was stored in a file. The results were then analyzed by the different parameters. Since the auxiliary property approach is not implemented for m:n relationships (cf. section 2.4), there is always one analysis for all approaches and 1:1/1:n relationships and another without the auxiliary property approach but for all relationship types. Figure 1 and 2 show the analysis results by SQL dialect with and without the auxiliary property.

Fig. 1: Average query duration, by SQL dialect, for 1:1 and 1:n relationships

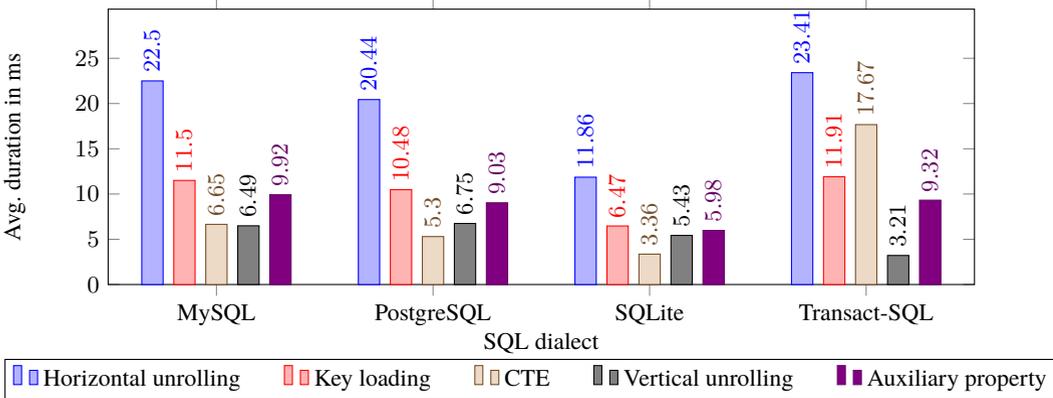
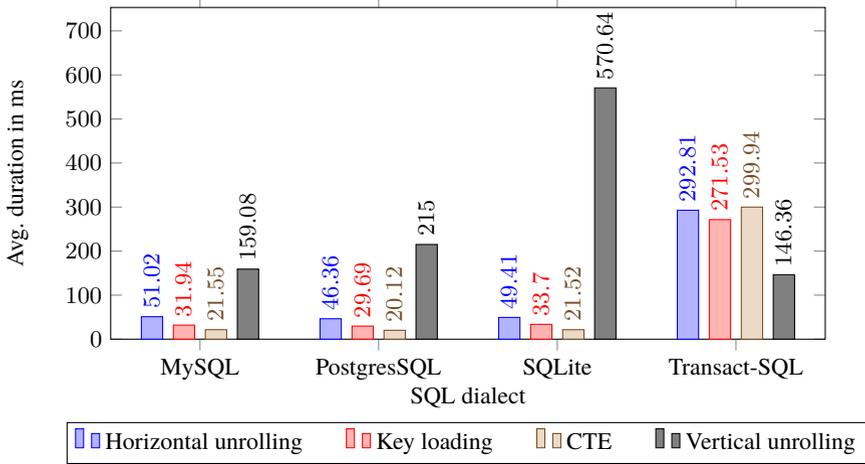


Fig. 2: Average query duration, by SQL dialect, for 1:1, 1:n and m:n relationships



The figures show that recursive CTEs are a lot slower when using Transact-SQL, as it was anticipated. They also show a pattern consistent across all results: When there is no join table (1:1 and 1:n relationships), and the recursion depth is low, vertical unrolling is quite fast. But vertical unrolling becomes very slow when there is a join table (m:n relationship), especially when using SQLite. The same goes for higher recursion depth, although the corresponding analysis is not shown here for brevity. This slowdown happens because the number of joins increases in both cases. Also, horizontal unrolling and queries with auxiliary property are usually slower than at least one other approach, while key loading is usually faster than those two methods but slower than recursive CTEs.

## 4 Conclusion and further research

Recursive CTEs are the fastest approach in many scenarios; even when they are not, they are usually not far behind. They also scale well with larger amounts of data. The same goes for key loading, but it is a bit slower. For these reasons, recursive CTEs are the most efficient solution, but depend on the used SQL dialect. They should not be used with Transact-SQL, because of the slow performance and the missing handling of cyclical data. One should use key loading instead if one wants to avoid these disadvantages. If smaller amounts of data and a low recursion depth can be assumed, one can also use vertical unrolling, since it performs well in these situations. The use of horizontal unrolling can not be recommended since there is no scenario where it is the fastest. Queries with auxiliary property are also not recommended since this approach can only be used in specific situations and is also slower than vertical unrolling in many of them. Further research could be done using further SQL dialects like PL/SQL, and/or using another ORM Framework. More extensive testing could also be done with higher recursion depths and more entities.

## References

- [BSW12] Boniewicz, A.; Stencel, K.; Wiśniewski, P.: Unrolling SQL: 1999 Recursive Queries. In (Kim, T.-h.; Ma, J.; Fang, W.-c.; Zhang, Y.; Cuzzocrea, A., eds.): Computer Applications for Database, Education and Ubiquitous Computing. Springer, 2012.
- [HSS18] Heuer, A.; Saake, G.; Sattler, K.-U.: Datenbanken: Konzepte und Sprachen. mitp, 2018.
- [Mi21] Microsoft: Entity Framework Core, 2021, URL: <https://learn.microsoft.com/en-us/ef/core/>, visited on: 12/15/2022.
- [Mi22a] Microsoft: Language Integrated Query (LINQ) (C#), 2022, URL: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>, visited on: 12/15/2022.
- [Mi22b] Microsoft: WITH common\_table\_expression (Transact-SQL), 2022, URL: <https://docs.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver16>, visited on: 12/15/2022.
- [Or22] Oracle: 13.2.15 WITH (Common Table Expressions), 2022, URL: <https://dev.mysql.com/doc/refman/8.0/en/with.html>, visited on: 12/15/2022.
- [Po22] PostgreSQL-Global-Development-Group: 7.8. WITH Queries (Common Table Expressions), 2022, URL: <https://www.postgresql.org/docs/14/queries-with.html>, visited on: 12/15/2022.
- [SQ22] SQLite-Team: The WITH Clause, 2022, URL: [https://www.sqlite.org/lang\\_with.html](https://www.sqlite.org/lang_with.html), visited on: 12/15/2022.
- [Sz11] Szumowska, A.; Boniewicz, A.; Burzańska, M.; Wiśniewski, P.: Hibernate the Recursive Queries - Defining the Recursive Queries Using Hibernate ORM. In (Eder, J.; Bielikova, M.; Tjoa, A. M., eds.): 15th East-European Conference on Advances in Databases and Information Systems. Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Toruń, Poland, 2011.



# Witness Generation for JSON Schema Patterns

Christoph Köhnen<sup>1</sup>

**Abstract:** JSON Schema is a schema language for the popular data exchange format JSON. This paper introduces an approach to convert regular expressions, which appear in ECMA-262 syntax in JSON Schema, into an alternative syntax, such that they may be compiled to finite-state automata. Specifically, we address the challenge that the ECMA-262 pattern syntax uses anchor symbols to mark the beginning and end of a word, which is not compatible with available libraries for automata manipulation. This is a step towards generating witnesses, i.e., JSON instances which are valid w.r.t. the given JSON Schema specification. We implement an algorithm proposed by Dominik Freydenberger to convert regular expressions into *brics* syntax. We show that we successfully address over 97% of the unique patterns found in a collection of thousands of JSON Schema specifications collected from GitHub.

## 1 Introduction

JSON Schema is a language for describing collections of JSON instances, where JSON is a widely adopted format for data exchange. This article describes a *student Bachelor thesis* project which targets a key problem in generating a *witness* for a given JSON Schema specification (short *schema*), which is a JSON instance valid w.r.t. this given schema. Witness generation has several important applications, such as checking schema containment [At22], a challenge also researched in [Ha21] for type-checking data science pipelines.

For example, consider the JSON Schema specification in Figure 1. Any witness must be of type object, as required in line 1. If the object has a property (line 2) whose name matches the pattern  $^(sur)?name\$$  (line 3), so "surname" or "name", then its value must be a string (line 4) matching the regular expression in line 5 with at least three characters (line 6). As this example illustrates, JSON Schema *patterns* can describe property keys (via `patternProperties`) or string-typed values (via the keyword `pattern`).

JSON Schema patterns are encoded in ECMA-262 syntax<sup>2</sup> and are actively used in practice: In analyzing a corpus of approx. 80K open source JSON Schema documents [Ba21], we found that 21% of the schemas contain patterns. Our approach cannot handle lookahead and lookbehind as well as the word boundaries `\b` and `\B`. The former one matches between a word and a non-word character without consuming one. But this concerns less than 3% of the unique patterns found in this corpus. We further found that the bulk of patterns actually describes regular languages. Some patterns in ECMA-262 can exceed the expressiveness of

---

<sup>1</sup> Universität Passau, Fakultät für Informatik und Mathematik, Lehrstuhl für Informatik mit Schwerpunkt Skalierbare Datenbanksysteme, Innstraße 33, 94032 Passau, Deutschland, koehne02@ads.uni-passau.de

<sup>2</sup> <https://json-schema.org/draft/2020-12/json-schema-core.html#rfc.section.6.4>

regular languages, due to backreferences [FS19]. We consider such occurrences as normal characters.

Translating the remaining patterns into finite-state-automata [HU79], we can generate string witnesses for patterns, simply by traversing a path from the initial state to some accepting state. From product automata, we can generate string witnesses that adhere to several constraints, e.g., matching *several* patterns as well as minimum and maximum lengths.

Our Java implementation of a tool for JSON Schema witness generation [At22] uses the automaton library `brics` [Mø17] for creating automata from regular expressions, as well as for computing automaton operations. However, `brics` relies on the syntax for regular expressions accustomed from computer science textbooks [HU79], assuming expressions to be *bounded*. Thus, the expression `bc*` matches "b" and "bc", but not "abcd". Yet in ECMA-262 syntax, the equivalent regular expression would have to be explicitly bounded as `^bc*$`.

```

1 { "type": "object",
2   "patternProperties": {
3     "(sur)?name$": {
4       "type": "string",
5       "pattern": "^[A-Z][a-z]*$",
6       "minLength": 3 }}}
```

Fig. 1: A JSON Schema specification.

**Contributions.** (1) We implement a novel algorithm to convert patterns from ECMA-262 to `brics` syntax. The conversion was suggested to us by Dominik Freydenberger. (2) We have integrated our implementation in a tool for JSON Schema witness generation [At22]. (3) We further present an empirical study on the applicability of our approach to patterns as they appear in tens of thousands of real-world schemas crawled from GitHub and provide a fully automated reproduction package of it. We can show that for 97% of the unique patterns found in this corpus, we can successfully apply our rewriting.

**Structure.** Section 2 introduces preliminaries and presents the conversion of regular expressions in ECMA-262 syntax to `brics` syntax. Section 3 presents and discusses our empirical evaluation. Section 4 reviews related work. Section 5 concludes.

## 2 Patterns and Witness Generation

```

expression ::= basicValue | object | array ;
basicValue ::= null | true | false | x | s ;
object ::= {k1 : J1, ..., kn : Jn} ;
array ::= [J1, ..., Jn] ;
```

Fig. 2: JSON grammar, adapted from [At22].

**Preliminaries.** The JavaScript Object Notation (JSON) is a data format where a *JSON document* (or *JSON expression*) has a syntax which is defined by the grammar in Figure 2, where  $n \geq 0$ ,  $x$  is a number,  $s$  is a string,  $J_1, \dots, J_n$  are JSON expressions, and all  $k_i$  are pairwise different key strings. JSON Schema also uses JSON syntax. A

formal semantics is defined in [At22; Pe16]. Specifically, we are interested in the keywords `patternProperties` and `pattern`, as illustrated in Figure 1.

|  |
|--|
| <pre> <i>expr</i> ::= (<i>expr</i> <b>alt</b>)? <i>seq</i> ; <i>seq</i> ::= (<i>char</i>   <i>group</i>   <i>qExpr</i>)* ; <i>qExpr</i> ::= (<i>char</i>   <i>group</i>) <i>quant</i> ; </pre> |
|--|

Fig. 3: JSON Schema pattern grammar.

**Approach.** We denote a regular expression extracted from a JSON Schema document as *JSON Schema pattern* (or short *pattern*). Since such a pattern follows the ECMA-262 syntax it can be defined by the grammar in

Figure 3 with the following tokens: **alt** is the alternation symbol |, *group* one of the alternatives (*expr*), (?:*expr*), (?<*str*>*expr*), (?!*expr*), (?=*expr*), (?<!*expr*), (?<=*expr*), *char* a character (or its unicode representation), a character class (like [abc], [a-z], [abcA-Z] or [^a-z]) or an anchor symbol (^ or \$), *str* a sequence of characters and *quant* a simple (+, \* or ?) or range quantifier ({m}, {m,n} or {m,},  $n \geq m \geq 0$ ).

The speciality of the ECMA-262 language is the use of the anchor symbols ^ for the beginning of a word and \$ for the end. The expression ^abc\$ in ECMA-262 syntax matches the string "abc" and nothing else while abc matches any string with "abc" inside, for example "012abcdef". To generate a string which matches a given regular expression in ECMA-262 syntax it can be helpful to create a finite-state automaton, since every regular language can be defined by such an automaton. The Java library `dk.brics [MØ17]` supports the creation of finite-state automata with regular expressions as well as the common operations of automata like concatenation, union, intersection or negation. Matching operations are also supported. Regular expressions are defined in the class `dk.brics.automaton.RegExp`.

A *regular expression in brics syntax* consists of the tokens listed in Figure 3, the symbols @ for any string and # for the empty language, but without anchor symbols, non- and named-capturing grouping and lookahead/-behind. It follows the same grammar with the exception that grouping only stands for simple grouping. Lookahead and lookbehind exceed the power of regular expressions. Freydenberger and Schmid worked that out in [FS19].

We now introduce an algorithm<sup>3</sup> to convert patterns from ECMA-262 to brics syntax, i.e. removes the anchor symbols, which works for nearly all patterns found in the corpus of JSON files from open-source projects on GitHub [Ba21]. To get rid of ^ resp. \$ we define the functions `h` and `nh` (short for `hat` and `nohat`) resp. `d` and `nd` (short for `dollar` and `nodollar`). `h` and `nh` compute regular expressions which do not use a symbol for the beginning of the word, `h( $\alpha$ )` matches the same language as the part of  $\alpha$ , where ^ is used, `nh( $\alpha$ )` the same language as the part of  $\alpha$ , where ^ is not used. `d` and `nd` compute regular expressions which do not use symbols for the beginning or end of the word, `d( $\beta$ )` matches the same language as the part of  $\beta$ , where \$ is used, `nd( $\beta$ )` the same language as the part of  $\beta$ , where \$ is not used. First we define a special predicate.

**Definition 2.1** *Let  $\alpha$  be a regular expression in ECMA-262 syntax. The predicate `nullable( $\alpha$ )` is true if and only if the empty word is contained in the language defined by  $\alpha$ .*

Examples for nullable patterns are `(^a$)*`, `(a|b?)` or `^$` which only accepts the empty word. Now we can define functions to compute the anchor and non-anchor parts of a pattern.

<sup>3</sup> Idea, algorithm, and replacement rules by Dr. Dominik D. Freydenberger (not published).

Tab. 1: Rules to compute the values  $h(\alpha)$ ,  $nh(\alpha)$ ,  $d(\alpha)$  and  $nd(\alpha)$  for a regular expression  $\alpha$  recursively.

| $\alpha$   | $h(\alpha)$  | $nh(\alpha)$                        | $d(\alpha)$  | $nd(\alpha)$                        |
|--|--|-------------------------------------|--|-------------------------------------|
| $\alpha \in \{\emptyset, @, \#\} \cup \text{CHAR}$ | #  | $\alpha$                            | #  | $\alpha$                            |
| $\wedge$   | $\emptyset$  | #                                   | #  | $\wedge$                            |
| $\$$   | #  | $\$$                                | $\emptyset$  | #                                   |
| $\alpha_1   \alpha_2$                              | $h(\alpha_1)   h(\alpha_2)$  | $nh(\alpha_1)   nh(\alpha_2)$       | $d(\alpha_1)   d(\alpha_2)$  | $nd(\alpha_1)   nd(\alpha_2)$       |
| $\tilde{\alpha}^+$                                 | $h(\tilde{\alpha}) \cdot nh(\tilde{\alpha})^*$                                 | $nh(\tilde{\alpha})^+$              | $nd(\tilde{\alpha})^* \cdot d(\tilde{\alpha})$                                 | $nd(\tilde{\alpha})^+$              |
| $\tilde{\alpha}^*$                                 | $h(\tilde{\alpha}) \cdot nh(\tilde{\alpha})^*$                                 | $nh(\tilde{\alpha})^*$              | $nd(\tilde{\alpha})^* \cdot d(\tilde{\alpha})$                                 | $nd(\tilde{\alpha})^*$              |
| $\tilde{\alpha}?$                                  | $h(\tilde{\alpha})$  | $nh(\tilde{\alpha})?$               | $d(\tilde{\alpha})$  | $nd(\tilde{\alpha})?$               |
| $\alpha_1 \alpha_2$                                | $h(\alpha_1) \cdot nh(\alpha_2)   h_{\alpha_1}^n(\alpha_2)$                    | $nh(\alpha_1) \cdot nh(\alpha_2)$   | $d_{\alpha_2}^n(\alpha_1)   nd(\alpha_1) \cdot d(\alpha_2)$                    | $nd(\alpha_1) \cdot nd(\alpha_2)$   |
| $\tilde{\alpha}^{\{m\}}$                           | $h(\tilde{\alpha}^{\{m,m\}})$  | $nh(\tilde{\alpha}^{\{m,m\}})$      | $d(\tilde{\alpha}^{\{m,m\}})$  | $nd(\tilde{\alpha}^{\{m,m\}})$      |
| $\tilde{\alpha}^{\{m,n\}}$ ( $m < 2, n > 0$ )      | $h(\tilde{\alpha}) \cdot nh(\tilde{\alpha})^{\{0,n-1\}}$                       | $nh(\tilde{\alpha})^{\{m,n\}}$      | $nd(\tilde{\alpha})^{\{0,n-1\}} \cdot d(\tilde{\alpha})$                       | $nd(\tilde{\alpha})^{\{m,n\}}$      |
| $\tilde{\alpha}^{\{m, \cdot\}}$ ( $m < 2$ )        | $h(\tilde{\alpha}) \cdot nh(\tilde{\alpha})^*$                                 | $nh(\tilde{\alpha})^{\{m, \cdot\}}$ | $nd(\tilde{\alpha})^* \cdot d(\tilde{\alpha})$                                 | $nd(\tilde{\alpha})^{\{m, \cdot\}}$ |
| $\tilde{\alpha}^{\{m,n\}}$ ( $m \geq 2, n > 0$ )   | $h(\tilde{\alpha} \cdot \tilde{\alpha} \cdot \tilde{\alpha}^{\{m-2, n-2\}})$   | $nh(\tilde{\alpha})^{\{m,n\}}$      | $d(\tilde{\alpha}^{\{m-2, n-2\}} \cdot \tilde{\alpha} \cdot \tilde{\alpha})$   | $nd(\tilde{\alpha})^{\{m,n\}}$      |
| $\tilde{\alpha}^{\{m, \cdot\}}$ ( $m \geq 2$ )     | $h(\tilde{\alpha} \cdot \tilde{\alpha} \cdot \tilde{\alpha}^{\{m-2, \cdot\}})$ | $nh(\tilde{\alpha})^{\{m, \cdot\}}$ | $d(\tilde{\alpha}^{\{m-2, \cdot\}} \cdot \tilde{\alpha} \cdot \tilde{\alpha})$ | $nd(\tilde{\alpha})^{\{m, \cdot\}}$ |

**Definition 2.2** Denote the set of all regular expressions by REGEXP and the set of all non-anchor character symbols and classes by CHAR. Let  $\alpha$  be a regular expression in ECMA-262 syntax. Define the functions  $h, nh : \text{REGEXP} \rightarrow \text{REGEXP}$  as follows. Let  $\tilde{\alpha}, \alpha_1, \alpha_2$  be regular expressions in ECMA-262 syntax and  $h_{\alpha_1}^n(\alpha_2)$  be  $h(\alpha_2)$  if  $\text{nullable}(\alpha_1)$  and # otherwise. Compute  $h(\alpha)$  and  $nh(\alpha)$  by applying recursively the rules from Table 1. Analogously, for a regular expression  $\beta$  in ECMA-262 syntax which does not use a symbol for the beginning of the word define the functions  $d, nd : \text{REGEXP} \rightarrow \text{REGEXP}$  as follows. Let  $\tilde{\beta}, \beta_1, \beta_2$  be regular expressions in ECMA-262 syntax which does not use a symbol for the beginning of the word and  $d_{\beta_2}^n(\beta_1)$  be  $d(\beta_1)$  if  $\text{nullable}(\beta_2)$  and # otherwise. Compute  $d(\beta)$  and  $nd(\beta)$  by applying recursively the rules from Table 1.

The rules in Table 1 are complete for all possible patterns in ECMA-262 syntax not containing word boundaries, lookahead or lookbehind since these features can match inside a pattern without consuming a character. Finally, we can formulate the algorithm.

**Algorithm 1** Algorithm to convert a regular expression from ECMA-262 to brics syntax.

**Input:** regular expression  $\alpha$  in ECMA-262 syntax without word boundaries, lookahead or lookbehind

1:  $\beta \leftarrow h(\alpha) | @ \cdot nh(\alpha)$  with  $h(\alpha)$  and  $nh(\alpha)$  computed using Definition 2.2.

2:  $\gamma \leftarrow d(\beta) | nd(\beta) \cdot @$  with  $d(\beta)$  and  $nd(\beta)$  computed using Definition 2.2.

**Output:** regular expression  $\gamma$  in brics syntax

**Example 2.3** For  $\alpha = (\wedge a\$ | b)?$  in ECMA-262 syntax we apply Algorithm 1. First, we remove  $\wedge$  by applying the rules from Table 1. We obtain

$$h(\alpha) = h((\wedge a\$ | b)?) = h(\wedge a\$ | b) = h(\wedge a\$) | h(b) = \underbrace{h(\wedge)}_{=#} \cdot \underbrace{nh(a\$)}_{=(\emptyset)} | \underbrace{h(a\$)}_{=#} = a\$,$$

$$nh(\alpha) = nh((\wedge a\$ | b)?) = nh(\wedge a\$ | b) = (nh(\wedge a\$) | nh(b)) = (\# | b) = b?$$

since  $\text{nh}(\hat{a}\$) = \text{nh}(\hat{\ }) \cdot \text{nh}(a\$) = \# \cdot \text{nh}(a\$) = \#$ ,  $\hat{a}$  is nullable and by using the shortcuts  $\text{h}(\tilde{\alpha}) = \#$  if  $\tilde{\alpha}$  does not contain a  $\hat{\ }$ ,  $\# \mid \tilde{\alpha} = \tilde{\alpha} \mid \# = \tilde{\alpha}$  and  $\text{C} \cdot \tilde{\alpha} = \tilde{\alpha}$ . We get  $\beta = a\$ \mid @ \cdot b?$ . Then we remove  $\$$  analogously by applying the rules from Table 1 and obtain

$$\begin{aligned} d(\beta) &= d(a\$) \mid d(@ \cdot b?) = d(a\$) \mid \# = d(a) \mid \text{nd}(a) \cdot d(\$) = \# \mid a \cdot \text{C} = a, \\ \text{nd}(\beta) &= \text{nd}(a) \cdot \text{nd}(\$) \mid \text{nd}(@) \cdot \text{nd}(b)? = a \cdot \# \mid @ \cdot b? = \# \mid @ \cdot b? = @ \cdot b? \end{aligned}$$

since  $\$$  is nullable and by using the shortcuts  $d(\tilde{\beta}) = \#$  if  $\tilde{\beta}$  does not contain a  $\$$ ,  $\# \mid \tilde{\beta} = \tilde{\beta} \mid \# = \tilde{\beta}$  and  $\tilde{\beta} \cdot \text{C} = \tilde{\beta}$ . The last step gives us the result  $\gamma = a \mid @ \cdot b? \cdot @$ , which is a regular expression in brics syntax that defines the same language as  $\alpha$  in ECMA-262 syntax.

### 3 Evaluation and Discussion

JSON Schema patterns do not only occur with a clause "pattern": `RegExp` but also as a pattern definition of a property name as mentioned in Section 2. Occurrences of both types are considered in the evaluation together. The numbers are based on a dataset of schemas found in open-source projects on GitHub [Ba21]. This corpus consists of 82,094 files. 17,747 of these files contain at least one pattern. This is 21.62%, so more than one fifth. Hence the goal to translate each JSON Schema pattern in an automaton-compatible syntax is relevant. After collecting all these patterns and eliminating duplicates we obtained 3,232 unique patterns. Our reproduction package is available at <https://doi.org/10.5281/zenodo.7586341>.

The implementation handles non- and named-capturing groups as capturing groups. In Table 2 we consider the numbers for syntactically invalid patterns and patterns for which we do not support a conversion. The former ones are 0.4% of the unique patterns and are mostly due to unescaped slashes, which are not allowed in ECMA-262. The latter ones are patterns containing word boundaries or lookahead, that is lookahead or lookbehind. These kind of patterns occurs in 2.44% of the unique patterns. These are the only possible cases for patterns which the algorithm cannot convert to brics syntax. For all the other ones, which is over 97% of the unique patterns, the procedure is successful.

In Table 3 we consider numbers for these brics-manageable patterns, where over 84% of them contain at least one anchor symbol, i.e.  $\hat{\ }$  or  $\$$ . However, in nearly all of these patterns the anchors are not inside the regular expression, which means that  $\hat{\ }$  resp.  $\$$  stands at the beginning resp. end of the expression. Only 0.67% of the brics-manageable patterns have anchors inside. If a pattern contains a non-nullable part before a starting or after an ending marker this pattern is unsatisfiable, i.e. it accepts the empty language. Fortunately, such patterns are scarce. Also nullable patterns, that are patterns which match the empty word (see Definition 2.1), are rare, they amount to less than 6% of the unique patterns.

As we can see in the experimental results, there are less than 3% of the unique patterns for which our approach fails. These are due to invalid patterns, word boundaries and lookahead. However, these kinds of patterns are also not supported by the former approach in [Ha21].

Tab. 2: Patterns extracted from the corpus.

|                        | Total | %      |
|------------------------|-------|--------|
| Unique patterns        | 3,232 | 100.00 |
| Invalid patterns       | 13    | 0.40   |
| Not supported patterns | 79    | 2.44   |
| Manageable in brics    | 3,140 | 97.15  |

Tab. 3: Patterns manageable in brics.

|                              | Total | %     |
|------------------------------|-------|-------|
| Patterns with anchors        | 2,648 | 84.33 |
| Anchors inside               | 21    | 0.67  |
| Patterns without anchors     | 492   | 15.67 |
| Nullable patterns (Def. 2.1) | 183   | 5.83  |

## 4 Related Work

Our implementation is integrated in a tool which can generate witnesses for JSON Schema documents [At22]. This tool can be used to check schema containment. An earlier approach to containment checking (not based on witness generation) is presented in [Ha21]. It also relies on an external automaton library, the Python greenery library <sup>4</sup>, which also uses a non-anchored syntax. Habib et al. unanchor the patterns from the schemas and unescape the anchor symbols before using greenery especially for computing intersections of two regular expressions. These steps are only executed on the string representation of the regular expression, without parsing its structure. Thus, there are instances when the approach by Habib et al. fails to preserve pattern semantics, e.g. for patterns containing anchors inside and not at its beginning or end – different from our well-principled approach.

## 5 Conclusion

We have successfully integrated our syntax conversion in a tool for JSON Schema witness generation. The experiments in [At22] reveal that the generation of automata from complex patterns in JSON Schema can cause severe performance problems. This motivates a range of follow-up research, for instance, caching of reoccurring patterns, or a lazy computation of automata for the purpose of witness generation.

**Acknowledgments.** This article describes the results of a bachelor thesis project at the *University of Passau*, supervised by Stefanie Scherzinger. I thank Dominik Freydenberger for suggesting the conversion algorithm. I further thank the authors of [At22], specifically Lyes Attouche, Mohamed Amine-Baazizi, Dario Colazzo, Giorgio Ghelli, and Dario Colazzo for guidance and support. This contribution was partly funded by *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation) grant #385808805.

<sup>4</sup> <https://github.com/qntm/greenery>

## References

- [At22] Attouche, L.; Baazizi, M.-a.; Colazzo, D.; Ghelli, G.; Sartiani, C.; Scherzinger, S.: Witness Generation for JSON Schema. *Proc. VLDB Endow.* 15/13, pp. 4002–4014, 2022, URL: <https://www.vldb.org/pvldb/vol15/p4002-sartiani.pdf>.
- [Ba21] Baazizi, M.-A.; Ghelli, G.; Colazzo, D.; Sartiani, C.; Scherzinger, S.: sdb-s-uni-p/json-schema-corpus: JSON Schema Corpus Release 07/2021, using the updated data from <https://github.com/sdb-s-uni-p/json-schema-corpus>, commit hash #79f808b, July 2021, URL: <https://doi.org/10.5281/zenodo.5141199>.
- [FS19] Freydenberger, D. D.; Schmid, M. L.: Deterministic regular expressions with back-references. In: *Journal of Computer and System Sciences.* 105, pp. 1–39, 2019, URL: <https://doi.org/10.1016/j.jcss.2019.04.001>.
- [Ha21] Habib, A.; Shinnar, A.; Hirzel, M.; Pradel, M.: Finding Data Compatibility Bugs with JSON Subschema Checking. In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2021*, pp. 620–632, 2021, URL: <https://doi.org/10.1145/3460319.3464796>.
- [HU79] Hopcroft, J. E.; Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.
- [Mø17] Møller, A.: dk.brics.automaton – Finite-State Automata and Regular Expressions for Java, version 1.12-1, 2017, URL: <http://www.brics.dk/automaton/>.
- [Pe16] Pezoa, F.; Reutter, J. L.; Suarez, F.; Ugarte, M.; Vrgoč, D.: Foundations of JSON Schema. In: *Proceedings of the 25th International Conference on World Wide Web. WWW '16*, pp. 263–273, 2016, URL: <https://doi.org/10.1145/2872427.2883029>.



# List of Authors

## A

Abdelmageed, Aly, 879  
Abedjan, Ziawasch, 367  
Ahn, Minseon, 751  
Algergawy, Alsayed, 879  
Alhomssi, Adnan, 259  
Auer, Sören, 367  
Aumiller, Dennis, 195

## B

Babalou, Samira, 951  
Bang, Tiemo, 641  
Baumstark, Alexander, 105, 797  
Bayer, Mirjam, 657  
Behr, Henriette, 535  
Beilschmidt, Christian, 837  
Benson, Lawrence, 757  
Bergmann, Rico, 283  
Binnig, Carsten, 157, 711, 995  
Bittkowski, Meik, 1049  
Bodensohn, Jan-Micha, 157  
Boehm, Matthias, 707, 815  
Boissier, Martin, 47  
Bornschlegl, Marco Xaver, 901  
Brass, Stefan, 607  
Broneske, David, 697  
Bruchhaus, Sebastian, 901  
Burrell, David, 943

## C

Chatziliadis, Xenofon, 943  
Christen, Peter, 485  
Christen, Victor, 439  
Christmann, Philipp, 579  
Conrad, Stefan, 595

Cordova, José Andrés, 595

## D

Damme, Patrick, 815  
Das, Pronaya Prosun, 981  
Drönnner, Johannes, 837  
Dunkelau, Jannik, 595  
Duong, Manh Khoi, 595

## E

Eichler, Annika, 1023  
Elkafrawy, Passent, 879  
Ellakwa, Susan F., 879  
El-Shaikh, Alex, 773  
Engelmann, Björn, 1009  
Eppinger, Florian, 93  
Esmailoghli, Mahdi, 367

## F

Faeskorn-Woyke, Heide, 621  
Fan, Jing, 195  
Fett, Johannes, 763  
Fey, Görschwin, 1023  
Fischer, Tim, 1069  
Focken, Mareike, 621  
Franke, Martin, 439  
Freitag, Michael, 235  
Fruth, Leon, 829

## G

Gertz, Michael, 195  
Geyer, Andreas, 751  
Ghazimatin, Azin, 633  
Göllner, Sabrina, 933  
Gomez, Kevin, 485  
Gradl, Tobias, 829

Graefe, Goetz, 27  
Grünhagen, Arne, 1023  
Gutberlet, Tim, 1091

## H

Habich, Dirk, 283, 697, 751, 763  
Hahn, Tobias, 729  
Hansen, Yorik Timo, 657  
Hartmann, Claudio, 283  
Hasler, Charlotte, 621  
Hatem, Shahenda, 879  
Hättasch, Benjamin, 157  
Haubenschild, Michael, 259  
Heinz, Florian, 687, 821  
Hemmje, Matthias, 901  
Henneberg, Justus, 665  
Henrich, Andreas, 701, 829  
Hertzschuch, Axel, 283  
Hinneburg, Alexander, 607  
Hübscher, Leonardo, 417  
Hurdelhey, Ben, 47

## I

Ilic, Ivan, 305  
Islam, Tamjidul, 719

## J

Jack, Thomas, 981  
Jegan, Robin, 829  
Jiang, Lan, 417  
Jibril, Muhammad Attahir, 105, 797  
Jörz, Simon, 131

## K

Kaoudi, Zoi, 535  
Karam, Naouel, 701  
Keller, Jüri, 1049  
Kemper, Alfons, 183  
Kerth, Alexander, 665

Kerzel, Dominik, 965  
Killisch, Benjamin Uwe, 1105  
Kipf, Andreas, 707  
Klan, Friederike, 851  
Kleest-Meißner, Sarah, 511  
Kleinsteuber, Erik, 951  
Kless, André, 621  
Klettke, Meike, 917  
Knolle, Harm, 621  
Kober, Urs, 763  
Koch, Maximilian, 367  
Köhnen, Christoph, 1113  
König-Ries, Birgitta, 701, 851, 879,  
951, 965, 1059  
Kosbü, Kimberley, 657  
Krause, Alexander, 751  
Kröger, Peer, 657  
Kudraß, Thomas, 1105  
Kulow, Andrea, 657

## L

Lambert, Jens, 621  
Langenecker, Sven, 995  
Laskowski, Lukas, 1099  
Lee, Dong Hun, 751  
Lehner, Wolfgang, 283, 751, 763  
Leis, Viktor, 259  
Lindenau, Arvid, 821  
Löffler, Felicitas, 851  
Luthra, Manisha, 707  
Lutsch, Adrian, 711

## M

Markl, Volker, 535, 943  
Mast, Marcel, 981  
Mattig, Michael, 837  
Mayerl, Maximilian, 555  
Medhat, Walaa, 879

Mertová, Lukrécia, 865  
Mogk, Ragnar, 711  
Moosleitner, Manfred, 221  
Müller, Wolfgang, 865  
Mundt, Martin, 711  
Mustafa, Tarek Al, 1059

## N

Naumann, Felix, 417  
Naumann, Lucas Fabian, 1083  
Neumann, Thomas, 183, 235

## O

Olijnyk, Andreas, 821  
Osterthun, Arne, 681

## P

Papenbrock, Thorsten, 391, 461  
Paradies, Marcus, 681  
Pensel, Lukas, 665  
Pietrzyk, Johannes, 751

## R

Rabl, Tilmann, 305, 757  
Rahm, Erhard, 439, 485  
Rakow, Thomas C., 621  
Reibert, Joshua, 681  
Reis, Thoralf, 901  
Restat, Valerie, 917  
Rey, Alice, 235  
Ritter, Daniel, 751  
Rohde, Florens, 439  
Rost, Christopher, 485  
Roy, Rishiraj Saha, 579

## S

Saatz, Inga Marina, 621  
Samuel, Sheeba, 965, 1059  
Sattler, Kai-Uwe, 105, 797

Sattler, Rebecca, 511  
Sauerbier, Janik, 1091  
Schaer, Philipp, 1009, 1049  
Schalles, Christian, 995  
Scheffler, Florian, 1105  
Schenkel, Ralf, 705  
Scherp, Ansgar, 705  
Schildgen, Johannes, 687, 821  
Schmid, Markus L., 511  
Schmidl, Sebastian, 391, 461  
Schuhknecht, Felix, 131, 665, 719  
Schüle, Maximilian E., 183, 931  
Schüll, Daniel, 729  
Schwarz, Christian, 763  
Schweikardt, Nicole, 511  
Schweitzer, Philip, 837  
Seeger, Bernhard, 701, 773, 837  
Seeger, Marcian, 391  
Shafiei, Fateme, 851  
Singh, Gagandeep, 711  
Sold, Florian, 1099  
Specht, Günther, 221, 555  
Störl, Uta, 93, 901, 917  
Sturm, Christoph, 995

## T

Teich, Jürgen, 729  
Tolovski, Ilin, 305  
Tropmann-Frick, Marina, 933, 1023

## U

Urban, Matthias, 157

## V

Vielhauer, Alexander, 391  
Vogel, Liane, 157  
Vötter, Michael, 555

## W

Wael, Tasneem, 879

Weidlich, Matthias, 511  
Weikum, Gerhard, 579  
Weisgut, Marcel, 47, 757  
Wenig, Phillip, 461  
Wiese, Lena, 981  
Wildermann, Stefan, 729

Witte, René, 851  
Wulff, Antje, 981

## **Z**

Zacharatou, Eleni Tzirita, 943  
Zangerle, Eva, 221, 555  
Zeuch, Steffen, 943