Experiences with Using a Pre-Trained Programming Language Model for Reverse Engineering Sequence Diagrams

Sandra Greiner[1], Nicolas Maier[2], Timo Kehrer[1]
[1]University of Bern, Switzerland
[2]University of Fribourg, Switzerland

Abstract *Reverse engineering software models from program source code has been extensively studied for decades. Still, most model-driven reverse engineering approaches cover only single programming languages and cannot be transferred to others easily. Large pre-trained AI transformer models which were trained on several programming languages promise to translate source code from one language into another (e.g., Java to Python). Thus, we fine-tuned such a pre-trained model (CodeT5) to extract sequence diagrams from Java code and examined whether it can perform the same task for Python without additional training.*

Motivation and Background. Reverse engineering aims at transforming source code to abstractions in the form of models, such as UML-like diagrams, which shall facilitate program comprehension and analysis, serving as starting point for optimizing and modernizing legacy software.

Existing model-driven reverse engineering tools are often trimmed to a single programming language, and they typically produce a structured, yet highly generic representation of the source code [3]. For instance, the well-known MoDisco tool [1] transforms Java source code (compatible to Java 1.6) into an instance of its own Java metamodel or the OMG's KDM metamodel[1]. While the Java metamodel abstracts from some details of the Java AST, it still represents the source file contents in a very fine-grained way, lacking the key feature of a model to focus on a dedicated aspect for a well-defined purpose. Both transferring the parser infrastructure to other source languages and extracting higher-level abstractions or domain-specific models require almost prohibitive manual effort.

These downsides call for more flexible reverse engineering tools which can be easily adapted to derive abstractions residing at different levels, and for a language-agnostic approach which can generalize reverse engineering tasks from a specific programming language to support distinct languages.

In sight of large pre-trained AI transformer models, such as GPT-3 and T5, smaller versions of these general-purpose natural language transformer models have been trained to learn different programming languages. These *pre-trained programming language models* (denoted as *AI models* in the sequel) are not restricted to perform analysis tasks in one programming language but promise to handle different ones. For instance, CodeT5 [4] cannot only summarize the behavior of a program's methods in natural language but also translate it into other programming languages. However, reverse engineering capabilities of AI models have not been explored yet [2]. In our preliminary study, we examined to what extent an AI model can perform the reverse engineering task of extracting sequence diagrams from methods written in different programming languages.

Reverse Engineering Task. As reverse engineering task, we desired to extract collaborations at the method-level of object-oriented software. For instance, Figure 1a depicts an excerpt of a Java method comprising three different method invocations, two of them being wrapped by conditional statements. Figure 1b depicts the corresponding sequence diagram that captures the deduced object collaborations.

The reason for eventually choosing this reverse engineering task is twofold. First, on top of Java parsing technology, such as MoDisco, the task can be scripted in a straightforward way, allowing us to easily generate sets of training and validation data. Second, since the input to the AI model is limited to a rather small set of tokens, we expect to not exceeded these limits by the size of the examined methods. In fact, we easily reached these limits in an earlier experiment in which we tried to use an AI model to transform an entire Java code base into an instance of MoDisco's Java metamodel.

Experimental Setup and Results. In our experiments[2], we employed CodeT5 (small[3]), an AI model which outperformed the previous state-of-the-art AI model, PLBART. CodeT5 is trained on seven programming languages (including Java and Python) to perform, for instance, code-to-code translation.

Java To Sequence Diagram For training the AI model on that task, we retrieved more than 350k methods from CodeSearchNet and ran the MoDisco discoverer on them. A script converts the result-

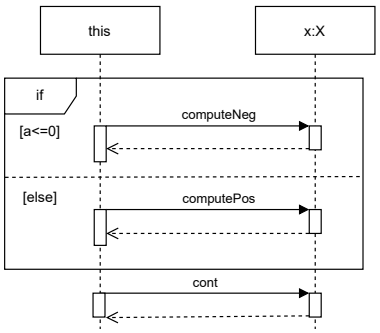
[1]https://www.omg.org/spec/KDM/1.4/About-KDM/

[2]https://doi.org/10.5281/zenodo.7628609

[3]https://huggingface.co/Salesforce/codet5-small

```java
void foo(int a) {
  // ...
  if (a <= 0) {
    x.computeNeg();
  }
  else {
    x.computePos();
  }
  x.cont();
}
```

(a) Java method

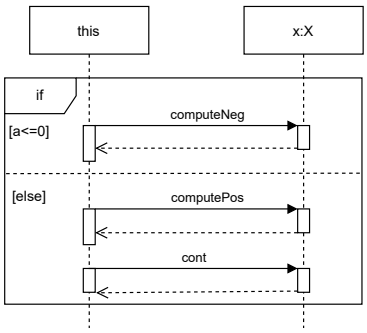


(b) Sequence diagram

Figure 1: Original Java source code and *correctly* reverse engineered sequence diagram.

```python
def foo(a):
  # ...
  if (a <= 0):
    x.computeNeg();
  else:
    x.computePos();
  x.cont();
}
```

(a) Python method



(b) Sequence diagram

Figure 2: Original Python source code and *incorrectly* reverse engineered sequence diagram.

ing XMI-files into JSON-files[4] comprising a custom description of a sequence diagram for each Java `MethodDeclaration`. For fine-tuning the AI model, we split the resulting data set into training, validation, and test sets, consisting of about 366k, 13k, and 21k examples, respectively, and fed them as tokens into the net. The fine-tuned model determined more than 98% of the sequence diagrams correctly.

Python To Sequence Diagram As our goal is to transfer the trained knowledge to other programming languages, we examined whether the fine-tuned model can extract correct sequence diagrams for Python, too. Due to the lack of a ground truth, we manually inspected a sample of selected Python methods. For example, Figure 2a depicts the method shown in Figure 1a defined in Python, and Figure 2b shows the sequence diagram obtained from our fine-tuned model. The figures demonstrate that the sequence diagram is determined incorrectly: The model associates the method invocation `cont()` with the else-branch. Fine-tuning might have caused the model to learn recognizing brackets as delimiters and contradicts our expectations on transfer learning.

Critical Discussion. We briefly summarize the major lessons learned from our experiments as follows:

Technical Limitation The limitations on the number of input tokens hinders exploring the contents of an entire class. As a consequence, the accuracy of the sequence diagram is limited as it may be unclear on which concrete objects the methods are invoked. However, given the knowledge of the input project and with additional fine-tuning of the task, we are confident that at least a post-processing mechanism can extract information about interacting objects and the corresponding classes.

Limited Transfer Learning The model transforms the Java methods into expected sequence diagrams almost perfectly (more than 98% accuracy and 99% CodeBLEU). Thus, fine-tuning the model according to the developers' recommendations worked as expected, and the aforementioned technical limitations are no serious issues.

Our initial results indicate that fine-tuning to convert Java methods may cause the AI model to forget knowledge about relationships among different pre-learned programming languages. If transfer learning between multiple programming languages is hindered by fine-tuning, it remains questionable whether the effort of training the language model pays off.

Conclusion and Outlook. All in all, we demonstrated how to reverse engineer sequence diagrams from Java methods using a pre-trained programming language model. While, after fine-tuning, the learned task was accomplished almost perfectly, our initial motivation of transferring the learning to another programming language did not meet our expectations. However, we did not experiment with different sizes for fine-tuning or with learning multi-tasks, yet. Therefore, future work may explore these two directions as well as the break-even point when the usage of an AI model exceeds conventional manual coding.

[4] Another script can transform the JSON-file into a vector graphic for visualization purposes.

References.

[1] Hugo Brunelière, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032, 2014.

[2] Changan Niu, Chuanyi Li, Bin Luo, and Vincent Ng. Deep learning meets software engineering: A survey on pre-trained models of source code. In *IJCAI-22*, pages 5546–5555. International Joint Conferences on Artificial Intelligence Organization, 2022.

[3] Claudia Raibulet, Francesca Arcelli Fontana, and Marco Zanoni. Model-driven reverse engineering approaches: A systematic literature review. *IEEE Access*, 5, 2017.

[4] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708. ACL, 2021.