# Exploration and systematic assessment of the resource efficiency of Machine Learning

Achim Guldner[1] Sandro Kreten[1] Stefan Naumann[1]

**Abstract:** Estimations of today's energy consumption of information and communication technologies (ICT) range from 2 to 9 % of the total produced energy and forecasts for the year 2030 predict an increase up to 21 %. Even though these numbers are controversial, it cannot be denied that the consumption growth of large impact factors, like data centers, networks, consumer devices, and the production of ICT needs to be reduced. In addition to Green IT, which is primarily focused on hardware, software is increasingly seen as an energy consumer with considerable savings potential. In this paper, we take a look at software for artificial intelligence (AI) and especially machine learning (ML). We describe a method for in-depth measurement and analyses of the energy consumption and hardware usage of ML algorithms and a series of experiments where we use the method on convolutional neural networks (CNN). We also compare existing estimation methods with our own. As outlook, we propose a holistic approach along the AI life cycle and additional experiments and assessments that could show potential efficiency improvements and consumption savings in AI.

**Keywords:** resource efficiency; energy and hardware assessment; measurement and analysis method

## 1 Introduction

In the last decade, AI and ML gained significant momentum, both in research and society. New technologies, like general-purpose computing on GPUs[2], new optimization algorithms, like ADAM (introduced by Kingma et al. in 2015 [KB15]), and new libraries, like TensorFlow (first released in 2015), brought the field to a level never seen before. Of course, this trend brings many advantages, especially, because of the many areas it can be applied to and the usability improvements that enable a more widespread usage. Companies such as Google, Amazon, and Microsoft already provide networks in their ML development environments, which can be applied in various productive areas. Thus, the combination of different pre-trained networks is often faster and more successful than the development of new processes. Especially in speech, text, and image recognition, rapid success can be achieved with pre-trained networks and ready-to-use data sets.

Recently, the implications of AI on a sustainable society shifted into research focus. Many works seek to implement new solutions to environmental issues, using machine learning.

[1] Hochschule Trier, Umwelt-Campus Birkenfeld, Postfach 1380, 55761 Birkenfeld, Germany {a.guldner|s.kreten|s.naumann}@umwelt-campus.de

[2] like Nvidia's Compute Unified Device Architecture (CUDA), released in 2007

Khakurel et al. [Kh18] find significant impacts of AI, both positive and negative, on all five dimensions of sustainability. Vinuesa et al. [Vi20] state that AI can enable the accomplishment of 134 targets across all of the United Nations 17 Sustainable Development Goals (SDGs), but it may also inhibit 59 targets. AI and ML can also be used in efforts to reduce the energy- and resource consumption across many subtopics, as evidenced, e.g. in the new journal "Energy and AI" (Jin et al. [JOJ]). The evaluation of the sustainability of AI and ML was first addressed by Strubell et al. [SGM19], estimating the $CO_2$ emissions of Natural Language Processing (NLP). They use Intel's RAPL tool[3] and nvidia-smi[4] while training the networks and approximate the energy usage and $CO_2$ emissions. They report, e.g. that a neural architecture search for translating English to German used 415409 kilowatt-hours (kWh), or 656347 kWh, factoring in Power Usage Effectiveness (this equals the annual energy consumption of ca. 200 households). Henderson et al. [He20] propose an *experiment-impact-tracker* framework for Python that also uses RAPL and nvidia-smi and encapsulates the assessment of the energy consumption. Schmidt et al. [Sc21] also provide a Python package based upon RAPL and nvidia-smi, called *CodeCarbon* that takes into account the computing infrastructure, location, usage, and running time. Both tools estimate how much $CO_2$ is produced by the algorithm. Canilang et al. [Ca21] take a look at the consumption in edge AI applications. This is especially advisable because of the limited hardware of those devices.

Since 2009, our research focuses on developing and assessing green and sustainable software systems. We defined *Sustainable Software* as software whose development, deployment, and usage results in minimal direct and indirect negative impacts or even positive impacts on the economy, society, human beings, and the environment (Naumann et al. [Na11]). From this definition, we derived criteria to evaluate a software product's sustainability in Kern et al. [Ke18], which includes our measurement method. In Naumann et al. [NGK21], we describe the process of the creation of a sustainability label (Blue Angel) for software products.

In this paper, we base the assessment of the sustainability of AI and ML upon the approach devised in these previous works and set three goals. We (1) assess, if the implemented measurement and analysis approach can be applied to the area of AI and ML and modify it where necessary, (2) perform two experiments and assess the energy consumption and hardware usage of a PC while training a CNN and compare it with the existing estimation approaches, and (3) provide an outlook on further experiments, assessments, and a holistic approach along the whole AI/ML life cycle.

## 2 Method

In this section, we describe the measurement setup and how we applied it to ML. The setup (depicted in fig. 1) is based upon Kern et al. [Ke18] and follows ISO/IEC 14756, as

[3] `https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl` [accessed 2021-04-04]
[4] `https://developer.nvidia.com/nvidia-system-management-interface` [accessed 2021-04-04]

introduced by Dirlewanger [Di06]. It consists of a system under test (SUT) on which the software that is to be assessed, is executed. In this case, the "software" is a Python script that trains CNNs. The power draw of the SUT is measured by a power meter (PM). A workload generator (WG) puts load on the SUT. With desktop or client-server software, this usually is a script that repeatedly performs and logs user inputs via an automation tool. Here, we used a python-script that executes and logs the training process. The recorded data is aggregated and evaluated (DAE) with a script, written in `R`, to produce the efficiency report.

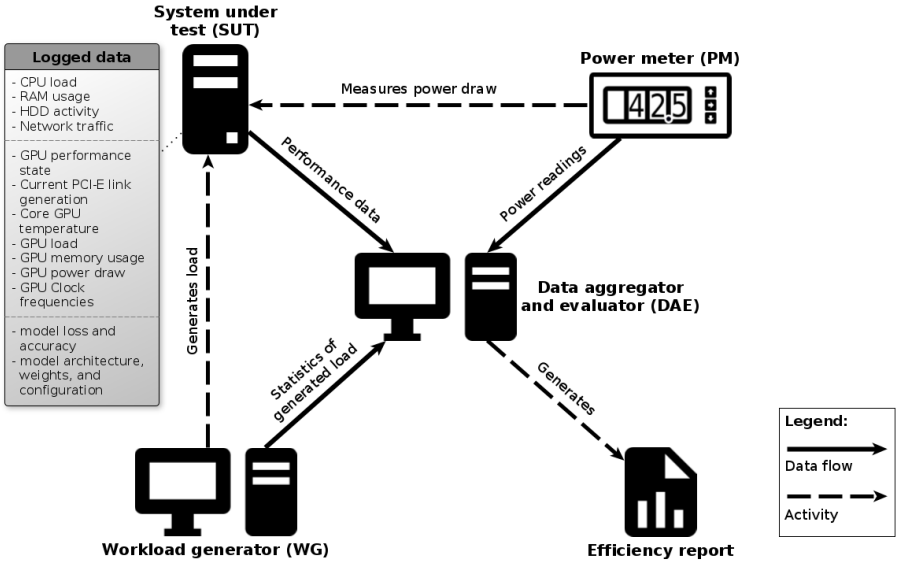


Fig. 1: Measurement setup

The PM saves the per second average power draw of the SUT. The WG script logs timestamps for the start and end of the test runs and training phases (epochs). The SUT collects its own performance data (CPU load, RAM usage, etc.) using collectl[5]. In addition to the previous setups, we integrated graphics card usage logs, collected with nvidia-smi and training and validation loss and accuracy in the analysis process. For later reference, the finished model architecture is also exported (using `model.save`).

The components of the used SUT are detailed in table 1. For the experiments, we set up the described software stack and measured the power draw and hardware usage for a baseline measurement (SUT runs only the operating system) and 3 usage scenarios: *idle* (Python script is running with no net being trained), *CNN* (training a CNN to classify pictures) and *transfer learning* (using a pre-trained CNN and training only the last layer). The scenarios are described in more detail in section 3. To combine and compare the method with the approaches from Henderson et al. [He20] and Schmidt et al. [Sc21], we set up additional

[5] `http://collectl.sourceforge.net/` [accessed 2021-04-09]

Tab. 1: SUT hardware specifications and software stack

| **Component** | **Specifications** |
|---|---|
| CPU | AMD Ryzen 7, 1700 |
| RAM | 16GB G.Skill RipJaws V DDR4-3200 DIMM |
| Main Board | MSI B350 PC MATE |
| Power supply | Cooler Master Silent Pro Gold, 600 W |
| SSD | 256GB WD Black M.2 2280 NVMe SSD (WDS256G1X0C) |
| Graphics card | ZOTAC GeForce GTX 1060 AMP! Edition, 3 GB 192 Bit GDDR5 |
| Operating System | Ubuntu 20.04.2 LTS |
| Python | Version 3.8.5 |
| TensorFlow | Version 2.4.1 |
| CUDA | Version 10.1.243 |

scenarios, where we measured the SUT's power draw, while simultaneously running the experiment-impact-tracker and CodeCarbon, respectively.

Finally, to get a feeling for the accuracy of nvidia-smi's gpu power draw readings, we compared the logging results with voltage and current measurements at the power plug of the graphics card while running GPU benchmarks. This is by no means as accurate as the measurement with the PM, but the results showed that the nvidia-smi logs are believable. The whole measurement setup is shown in fig. 2.

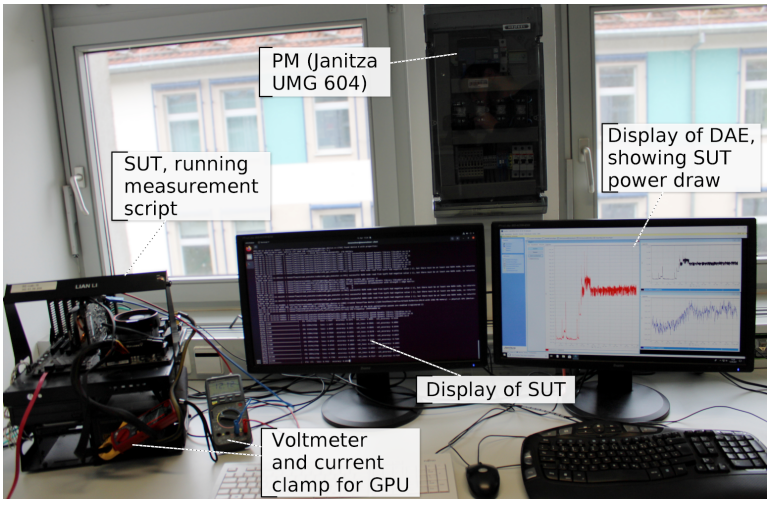


Fig. 2: Photo of measurement setup

The analysis method is largely identical to the one described in [Ke18]. We average each logged hardware component over the test runs and calculate the average energy consumption of the scenario in watt-hours [Wh], as the sum over all per-second averages of the power

draw (recorded by the PM during the execution of the scenarios, resulting in watt-seconds) and dividing the result by $3600\frac{s}{h}$. Subtracting the mean baseline values, adjusted for the scenario duration, gives us the part of the energy consumption and hardware usage that is *induced by the software*. Additionally, we calculate these indicators also for each training epoch.

In order to make the experiments and analysis reproducible, we provide all source code files, measurement data, analysis scripts, and efficiency reports in a replication package as git repository at `https://gitlab.rlp.net/a.guldner855830/exploration-and-systematic-assessment-of-the-sustainability-of-machine-learning/-/tree/master`[6].

## 3 Experiments

As a first set of experiments, we used the method to assess the energy consumption and hardware usage of the SUT while training CNNs to classify images from the `tf_flowers` data set[7]. The WG script first groups the images into folders according to their labels (roses, daises, dandelions, sunflowers, and tulips) and then randomly augments them (re-scale, rotate, change width- and height, flip horizontally, and zoom) to enhance the training process and have access to more training and validation data sets. Figure 3 depicts some examples. Finally, the images are fed to the `fit` function using an `ImageDataGenerator` object.

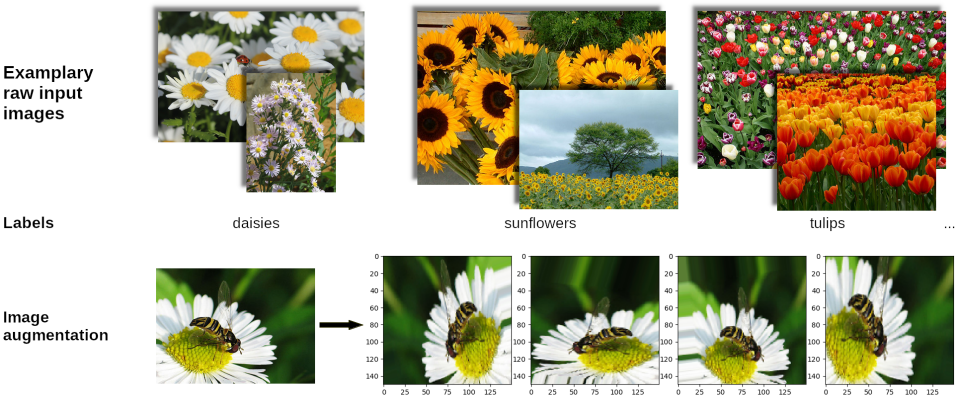


Fig. 3: Exemplary training images from the tf_flowers dataset (top) and image augmentation (bottom)

The scripts are based upon examples available from the TensorFlow git[8]. We ran each scenario (*idle*, *CNN*, and *transfer learning*) 30 times (="test runs"), so as to produce normally distributed samples [Ke18]. The scripts execute the scenarios described below, and log the time for the beginning and end of each test run. In addition to the scenarios, we

[6] the model architectures are available at `https://seafile.rlp.net/d/6001c483b42342a0bdb5/`
[7] available at `https://www.tensorflow.org/datasets/catalog/tf_flowers` [accessed 2021-04-09]
[8] `https://github.com/tensorflow/examples` [accessed 2021-04-09]

also recorded the baseline for the SUT, with only the operating system installed and without any scripts running. The idle scenario consists of the same script as the CNN and transfer learning scenario, except that instead of a model being trained, the execution is suspended for 10 minutes, using Python's `time.sleep()` function. This was included e.g. to simulate developers running an interactive session when implementing algorithms.

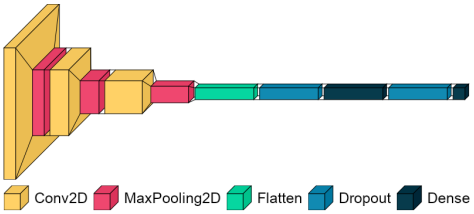


Fig. 4: CNN scenario model visualization

The CNN scenario consists of a `Sequential` model with 3 sets of `Conv2D` and `MaxPooling2D` layers, followed by a `Dense` hidden layer with 512 nodes and an output layer with 5 nodes (see fig. 4)[9]. For the transfer learning scenario, *MobileNet V2* from TensorFlow Hub is used[10]. Here, only the last layer is exchanged for an output layer with 5 nodes and only this layer is trained (this method is called "fine-tuning"). Both nets use the `adam` optimizer for a maximum of 100 training epochs, or until a validation accuracy of 80 % (CNN scenario) or 85 % (transfer learning scenario) is reached. This usually results in a different number of epochs for each test run. Therefore, the scripts also log timestamps for each *action* of the training process: image preparation, creation of the net, training process up to 80 % or 85 % validation accuracy, and for each epoch. After the test run, the script exports the trained model architecture and history (training- and validation-accuracy and loss) and resets the conditions as they were before the test run (deleting folder structures and pre-trained models). The actions within each test run are structured as follows:

```
test run 1
    prepare images (unzip data set, sort into folders, create ImageDataGenerator)
    create CNN (build model layers (CNN scenario) or download and prepare
    pre-trained model (transfer learning scenario), compile model)
    train CNN until set validation accuracy is reached
        epoch 1
        epoch 2
        ...
        epoch n
    export and clean-up (not part of the measurement)
test run 2
...
```

[9] Created with visualkeras, available at `https://github.com/paulgavrikov/visualkeras/` [accessed 2021-04-13]
[10] `https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4` [accessed 2021-04-09]

## 4 Analysis results

We now present an overview of the analysis and some explorative aspects. Table 2 shows the recorded data. From it, we calculated the energy consumption for the SUT and the graphics card, as well as the hardware usage averages, as described in section 2. Because the test run durations vary, we also calculate the averages for the whole training process and for each epoch. The data is visualised and aggregated in the report documents in the replication package.

Tab. 2: Recorded data, units, and recording method

| **Indicator** | **Unit** | **Method** |
|---|---|---|
| SUT power draw | W | Power meter |
| SUT total CPU load | % | collectl |
| SUT RAM usage | kB | collectl |
| SUT disk activity (reading and writing) | kB | collectl |
| SUT network traffic (sending und receiving) | kB | collectl |
| GPU performance state | no unit | nvidia-smi |
| GPU PCIe link generation | no unit | nvidia-smi |
| GPU core temperature | °C | nvidia-smi |
| GPU load | % | nvidia-smi |
| GPU memory load | % | nvidia-smi |
| GPU memory allocated | MiB | nvidia-smi |
| GPU power draw | W | nvidia-smi |
| GPU SM, memory and graphics clock frequency | MHz | nvidia-smi |

As an overview, fig. 5 shows the per second and overall average SUT power draw for the two usage scenarios. The results show the expected large benefits of using a pre-trained net and fine tuning it: The energy consumption induced by the transfer scenario amounts only to 0.374 Wh, compared to 10.032 Wh for the model that is trained from scratch (see fig. 6). Of course, this does not take into account the energy that was used to train the MobileNet in the first place. Thus, the two scenarios are not directly comparable, but that was also not the intention in this paper.

Looking at epoch-level, it is interesting to note that the energy consumption for the training process in the transfer scenario used 0.448 Wh per epoch with an average duration of 29.95 seconds, whereas for the CNN scenario, each epoch used only 0.423 Wh and, on average, took 28.304 seconds. This is even more surprising, because the complexity of the self-trained CNN is much larger (total and trainable parameters according to `model.summary()`: 25,716,773) than that of the adapted MobileNet (total parameters: 2,264,389; trainable parameters: 6,405).

Because the reports provide an in-depth analysis down to epoch-level for all recorded indicators, depending on the use case, it is possible, e.g. to analyse the energy consumption and hardware usage up to a certain epoch or until the validation accuracy of the net reaches

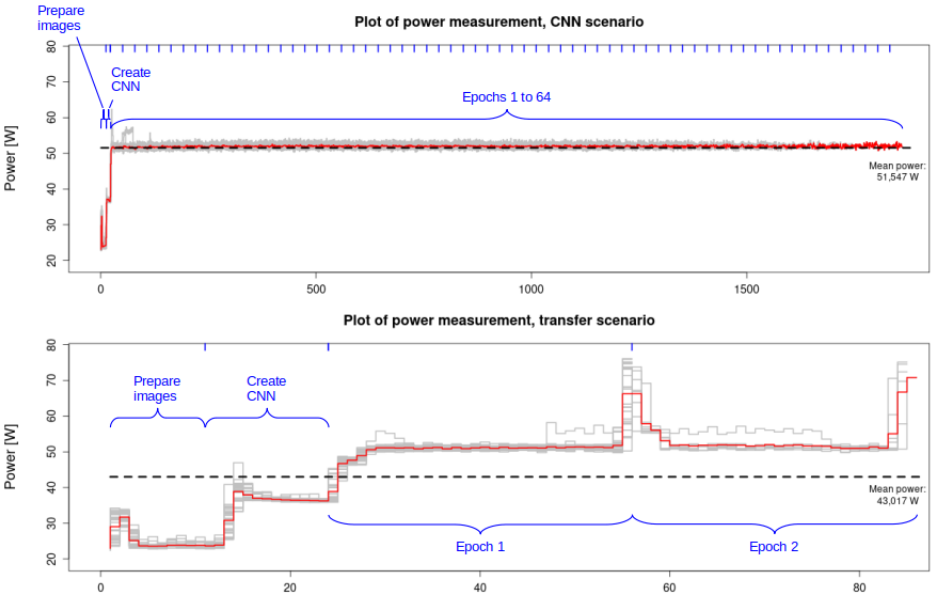


Fig. 5: Power draw measured with the PM for the CNN- (top) and transfer scenario (bottom).

a certain percentage. This allows detailed insights into how adjustments to the net relate to the energy and hardware consumption. E.g. it caught our eye that as soon as the model is initialized with `model = Sequential()`, the GPU memory was allocated even though no layers had been added or trained. Here, more research is necessary (see also section 5).

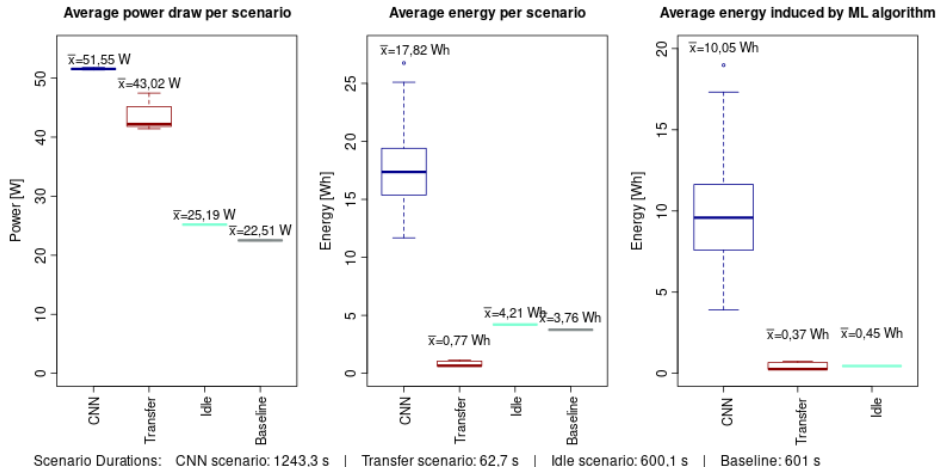


Fig. 6: Power draw and energy usage overview for all scenarios.

Fig. 6 depicts the average power draw, the average energy consumption, and the average energy consumption induced by the ML-algorithm (scenario energy minus adjusted baseline energy), per scenario. From this overview we can gain some insights: (1) considering energy consumption, using a pre-trained CNN seems to be by far the better choice for image classification, (2) the GPU's energy consumption rises from about 35 % in the baseline to about 46 % in the idle scenario and to about 70 % in the scenarios that train CNNs.

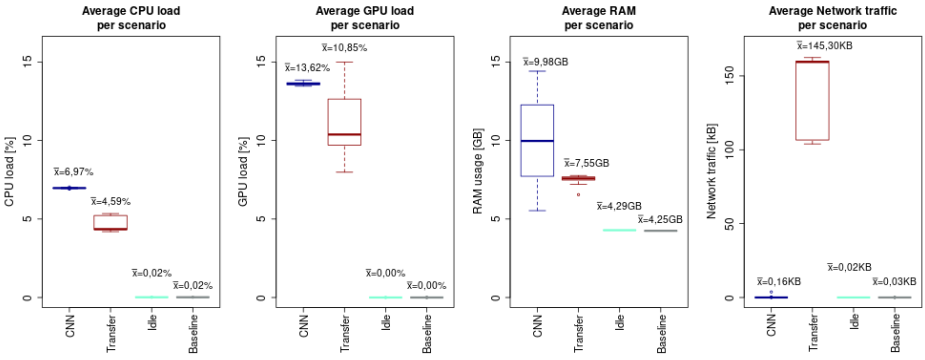


Fig. 7: Hardware usage overview.

For fig. 7, we chose to plot the CPU and GPU load, RAM usage, and network traffic as the main hardware usage indicators. Here, we find that the average hardware usage is also, for the most part, lower for the transfer scenario than for the CNN scenario. However, because the pre-trained model needs to be downloaded for the transfer scenario, this generates some network traffic and disk activity.

Another point we wanted to investigate was if the loss or accuracy would correlate with the energy consumption. This was, however, not the case in our measurements. What we did find was that the energy consumption per epoch correlates with the GPU temperature ($r = 0.98$), as can be seen in fig. 8. Likely, the GPU draws more power when it heats up.

From the collected data, it is also possible to calculate an *efficiency factor*, based upon the metrics proposed by Johann et al. [Jo12] as

$$\text{Energy efficiency} = \frac{\text{Useful work done}}{\text{Used energy}}.$$

For this paper, we used the validation accuracy as the "useful work done" in the training process. This results in the unit *validation accuracy per watt-hour*. It is depicted in fig. 8. Because the accuracy is not a sufficient measure for the "work" of the neural net, in the future more indicators should be taken into account (see section 5).

In summary, it seems advisable to include at least the following indicators for further analyses: Total energy consumption, GPU and CPU energy consumption (if available), CPU and GPU load, RAM and GRAM usage, network traffic and disk activity, as well as loss and

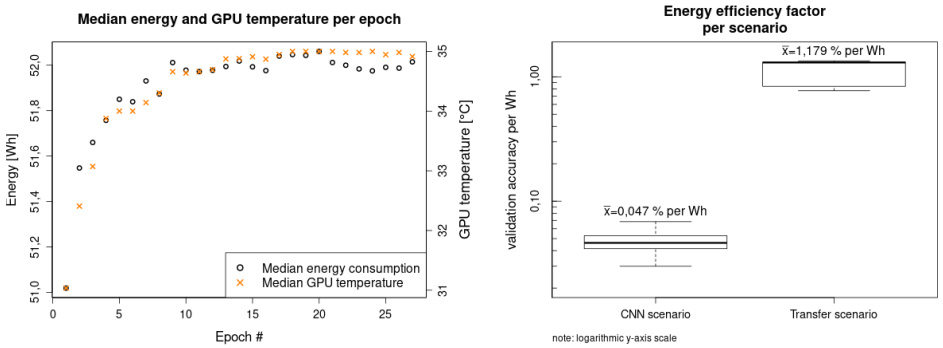


Fig. 8: Energy consumption per CNN scenario epoch (left) and efficiency factor comparison (right).

accuracy. The baseline measurement also proves useful to assess the energy consumption and hardware usage that is induced by the algorithms.

## 5 Discussion and outlook

In this section, we discuss the goals set in section 1: Applying our approach to the field of AI/ML, comparing our method to the existing approaches, and providing an outlook on our planed work. The devised method seems promising for the systematic and in-depth assessment of the sustainability of ML algorithms. Where the trackers, especially CodeCarbon, provide a good tool for familiarizing oneself with the energy demands of the implemented algorithms, using a PM makes it possible not only to estimate the energy consumption, but to measure it. Combined with hardware usage logs, the calculation of efficiency factors, and baseline measurements to assess the consumption induced by the ML algorithms, the method allows comparing scenarios, down to source code level. This allows further experiments to assess how changes to the algorithms effect the resource efficiency.

Since this is the first evaluation of our method in the area of ML, there are some threats to validity that we now discuss. We used only one ML library (TensorFlow), one SUT, and one programming language (Python). To mitigate this, our scenarios were based on standard use cases and, as future work, we plan to extensively evaluate the approach on a large set of ML applications, ranging different languages and technologies. We took precautions to minimize side effects in the measurements, performed baseline measurements of the SUT energy consumption, and executed each test case multiple times in order to ensure the statistical relevance of the data. We assessed the precision of the measurement setup by calculating the standard deviation of the power measurements and execution times and checked the plausibility of the gpu measurements. There is still potential for measurement- and coding errors and conclusions may be prone to subjective interpretations. In order to

mitigate these threats, we jointly inspected and discussed the results in order to identify potential occurrences of divergent interpretations and provide a replication package.

For the comparison with “CodeCarbon” (CC) [Sc21] and “experiment-impact-tracker” (EIT) [He20], we wrapped the test runs with the `EmissionsTracker` from CC and the `ImpactTracker` from EIT, respectively and then executed them as before, while recording the same three scenarios again (see fig. 9). Baseline measurements cannot be recorded with the trackers, because they require python to be running (= idle scenario). Of course, since our SUT contains an AMD CPU, RAPL could not be used and only the graphics card was available to the trackers. The CC tracker worked as expected and reported approx. 78 % of the PM measurements, which is close to the nvidia-smi readings.

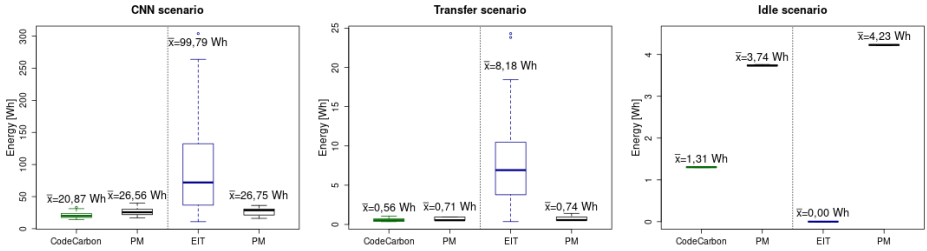


Fig. 9: Comparison of measured energy consumption of CC, EIT and our PM.

Although we used the EIT as described in the documentation with standard settings, the results cannot be correct. It is not the focus of this work to go into detail on why these errors occurred. From what we could see in a brief code review of EIT, the error is likely due to the fact that the power and energy averages are not calculated over time, but the first result from nvidia-smi over a measurement interval is used. This can produce large measurement errors because power spikes or drops can be missed. Furthermore, the tracker did not record any consumption for the idle scenario. This is the case, because the tracker considers the gpu-processes and bases the estimates on those. Since there are no processes running in the idle scenario, the tracker records no consumption. Compared to CC, EIT itself induces approx. 0.49 Wh additional energy consumption in the idle scenario, likely because it produces frequent console outputs and large amounts of log files with the standard settings. This results in additional power draw.

Since Strubell et al. [SGM19] do not provide or use a tool, but rather directly log the data with nvidia-smi and RAPL. Thus, the method seems directly comparable and should lead to similar estimation results if the scenarios we defined here were used. The focus here was on bringing the issue to the attention and quantifying the approximate financial and environmental costs. With our approach, this can be taken one step further, showing how adjustments to the net relate to energy and hardware consumption.

In the future, we plan to extend the method in two directions: influence factors along the AI life cycle and further comparison approaches. So far, only the training process of ML

algorithms was considered. We propose to stretch these considerations to a holistic approach along the complete life cycle of AI systems. This starts with data collection, taking sensors, networks, data reduction, etc. into account, which may influence the algorithms and their resource usage. The way the data is pre-processed and managed also needs to be addressed. This includes questions of data quality, storage, the use of synthetic data, data augmentation, the proportion of training, validation, and test data, etc. In the usage and optimization phase, influences from different classification, detection, and synthesis applications should be addressed, as well as possible saving that can be achieved with edge-AI. Furthermore, we plan to investigate the balance between the energy consumption during the training and usage phase. E.g. if a model is used only infrequently or over a short time, maybe it does not need high accuracy levels and thus shorter training, etc.

When modelling the algorithms, we plan to compare optimization algorithms and ML-libraries (TensorFlow, Theano, PyTorch, Keras, OpenCV, Caffee2, scikit, MXNet, etc.) with each other and with minimal implementations. Different ML-algorithms that can perform the same or similar tasks should be compared, as well as, implementations in different programming languages. In the area of deep learning, we plan to evaluate impacts from factors like network topology (e.g. layer structure, layer sizes, dropout layers), activation functions, gradient descent algorithms, performance measures, epoch and batch sizes, etc. For CNNs especially, the influence of factors like kernels, strides, pool sizes, pooling methods, number of pooling layers, number of filters, etc. should be assessed through comparison experiments.

Finally, the relation between energy consumption and ML success indicators needs to be evaluated, depending on the "useful work" of the ML-model for the energy efficiency factor. Here, measures like *precision*, *recall*, or *F-measure* during tests should be taken into account. This would result in e.g. the unit *correct classifications per Wh*.

## References

[Ca21] Canilang, H. M.; Caliwag, A.; Kwon, J.; Lim, W.: DNN Power and Energy Consumption Analysis of Edge AI Devices. In: KICS Winter Conference. 2021.

[Di06] Dirlewanger, W.: Measurement and Rating of Computer Systems Performance and of Software Efficiency. Kassel Univ. Press, Kassel, 2006.

[He20] Henderson, P.; Hu, J.; Romoff, J.; Brunskill, E.; Jurafsky, D.; Pineau, J.: Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. Journal of Machine Learning Research/, 2020.

[Jo12] Johann, T.; Dick, M.; Kern, E.; Naumann, S.: How to Measure Energy-Efficiency of Software: Metrics and Measurement Results. In: 1. Intl. Workshop on Green and Sustainable Software (GREENS), Zurich, Switzerland. Pp. 51–54, 2012.

[JOJ] Jin, D.; Ocone, R.; Jiao, K.: Energy and AI, Journal, URL: https://www.journals.elsevier.com/energy-and-ai/.

[KB15] Kingma, D. P.; Ba, J.: Adam: A Method for Stochastic Optimization, 2015.

[Ke18] Kern, E.; Hilty, L. M.; Guldner, A.; Maksimov, Y. V.; Filler, A.; Gröger, J.; Naumann, S.: Sustainable software products - Towards assessment criteria for resource and energy efficiency. FGCS 86/, pp. 199–210, 2018.

[Kh18] Khakurel, J.; Penzenstadler, B.; Porras, J.; Knutas, A.; Zhang, W.: The Rise of Artificial Intelligence under the Lens of Sustainability. Technologies 6/4, 2018.

[Na11] Naumann, S.; Dick, M.; Kern, E.; Johann, T.: The greensoft model: A reference model for green and sustainable software and its engineering. Sustainable Computing: Informatics and Systems 1/4, pp. 294–304, 2011.

[NGK21] Naumann, S.; Guldner, A.; Kern, E.: The Eco-label Blue Angel for Software - Development and Components. In: Advances and New Trends in Environmental Informatics. Progress in IS, Springer, pp. 79–89, 2021.

[Sc21] Schmidt, V.; Goyal, K.; Joshi, A.; Feld, B.; Conell, L.; Laskaris, N.; Blank, D.; Wilson, J.; Friedler, S.; Luccioni, S.: CodeCarbon, `https://github.com/mlco2/codecarbon`, 2021.

[SGM19] Strubell, E.; Ganesh, A.; McCallum, A.: Energy and Policy Considerations for Deep Learning in NLP. In: 57th Annual Meeting of the Association for Computational Linguistics. Pp. 3645–3650, 2019.

[Vi20] Vinuesa, R.; Azizpour, H.; Leite, I.; Balaam, M.; Dignum, V.; Domisch, S.; Felländer, A.; Langhans, S. D.; Tegmark, M.; Nerini, F. F.: The role of artificial intelligence in achieving the Sustainable Development Goals. Nature Communications 11/1, 2020.