Hardware Agnostic Energy Benchmarking For Machine Learning

Timo Laudi[1,2], Rolf Drechsler[1,3]
[1] Institute of Computer Science, University of Bremen, Germany
[2] Data Science Center of the University of Bremen, Germany
[3] Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{tlaudi,drechsler}@uni-bremen.de

Abstract

While energy consumption of Machine Learning (ML) applications is on the rise, understanding the impact of both hard- and software components on runtime efficiency remains a weak point in the research space. We present a hardware-agnostic approach for profiling and characterizing the energy consumption of ML applications leveraging the Open Neural Network Exchange (ONNX) format and ecosystem, demonstrate the portability of the approach over multiple hardware architectures, and highlight its potential to generalize to multiple manufacturers. We further showcase a first proof-of-concept demonstrating the necessity of clean and comparable training data across all relevant hardware platforms for the development of accurate energy consumption models.

1 Introduction

The use of ML has seen a huge rise in the past 20 years [1]. While methods for these ML-techniques have been around since the 1980s and even earlier [2, 3, 4] the increase of affordable computational capacity, e.g., in the form of Graphics Processing Units (GPU), has made these methods increasingly popular [5].

This allowed the development of models which are much more accurate and fast, but also much more complex and resource-intensive to run, in training as well as inference. The rise of model complexity and demand from the public has made energy consumption a significant concern for tech companies, with some planning to build or reopen nuclear power plants to support their datacenters. [6]

Much of the increase in computational capacity is due to the realization that GPUs lend themselves very well to the processing of vector data, which is central to the workloads encountered in ML, and the continued development and usage of GPUs towards this purpose. The specifics of hardware features and architecture in these GPUs have an extensive impact on performance and efficiency of models, and understanding this relationship, as well as choosing the right device from all available options, is crucial to creating energy-efficient and sustainable artificial intelligence.

Since GPUs of different manufacturers and architecture families can have very different feature sets, data interpolation across these boundaries is not feasible. Instead, developing a basis for this understanding requires reliable and comparable data across the entire landscape of GPU hardware.

These data can then be used to train reliable prediction models for energy consumption based on the chosen hardware architecture and features, as well as the problem at hand. In turn, a valid prediction will allow a valid decision-making process for choosing the right hardware for expected tasks.

Our contributions for this work are the following: First, we present a reliable benchmarking setup that ensures (a) tracking only the task and architecture dependent energy-consumption, and (b) is easily adapted to accommodate numerous hardware vendors and architectures. Second, we demonstrate our assumption, that generalizing insights gathered on one hardware platform may mislead models estimating energy consumption on another, with a first proof-of-concept tested on Generalized Matrix Multiplication (GEMM), which is a prominent task in many ML computations.

The remainder of this paper is structured as follows: Section 2 will list relevant related work and introduce some necessary background. Section 3 will introduce our methodology, demonstrating our general workflow and approach. Our experimental setup and relevant details are described in Section 4, whereas Sections 5 and 6 describe and discuss our results, respectively. Finally, Section 7 concludes this paper.

2 Related Work

In literature, there are multiple approaches of measuring and estimating the energy consumption of a given workload. The authors of [7] divide these approaches into External Power Meters and Internal Power Sensors. External meters are considered the most accurate, but require physical access to the device under test. In [8], the authors present a suite of micro benchmarks and evaluate them by attaching a physical power monitor to the GPU power supply. Instead, a power meter can also be attached to the wall outlet. As the authors of [9] point out, this approach yields an accurate characterization of the system-wide power consumption, but fails to generate data at a more fine-grained component level.

Internal sensors usually provide lower sampling rates and less accurate readings. According to [9], there is often little to no information about the exact way these sensors gather data or their exact accuracy. The advantage of this approach is, that data can often be conveniently queried using software provided by the manufacturer. This variant also enables the integration of power readings into automated test scripts. In [10, 11], the authors gather training data for an ML-based power prediction model using the *nvidia-smi* tool.

In [12], the authors add a third category of estimation models to the taxonomy, which consider indirect data, which are either metrics measured by hard- or software components or properties of the workload in question, and estimate the corresponding power draw or energy consumption. These models, in turn, are then divided into two subcategories.

On the one hand, there are analytical estimation models, which aim to estimate the energy consumption based on characteristics of the workload and the hardware, while considering the technical architecture of the system. For example, the authors of [13] model the memory of a GPU as a series of cache levels with different characteristics. They then propose a model to estimate a lower bound for the energy consumption of a workload based on the number of accesses per level and the sizes of the different caches. These models can yield valuable insights, but require a detailed understanding of the hardware in question, which becomes exceedingly complex in the face of diverse architectures.

On the other hand, there are data-based estimation models, which aim to learn relationships directly from datasets. These involve different methods of statistical modelling or deep learning. In [14], the authors evaluate multiple statistical methods, including linear regression, k-nearest-neighbors, and random forests, to estimate the power consumption of workloads at a given operating frequency. The authors of [15] make use of an artificial neural network and show that it can significantly outperform a multiple linear regression model.

As described in [9], total power consumption is usually seen as the sum of static and dynamic power consumption, where static consumption is the power consumed by the system in an idle state, while dynamic consumption is defined as the additional consumption induced by a workload and is usually derived by subtracting the static consumption from the total. While a direct measurement can only show the total power draw, the static power can reasonably be estimated by observing the system in an idle state.

Many power models make use of hardware provided Performance Counters (PC) profiled during execution to characterize workloads [13, 15]. According to [9], this has been a leading method. However, since PCs are specific not only to GPU vendors, but also architecture families, generalizing this method to many platforms can be challenging. In [11], the authors propose a candidacy algorithm, which automatically selects the PCs with the highest correlation to the energy consumption and demonstrate it on NVIDIA's Fermi and Kepler architectures. The algorithm involves a calibration phase to determine the impact of each PC, which suffers from large overhead of the profiling tool `nvprof`.

A slightly more high-level approach is the examination of the Parallel Thread Execution (PTX) assembly code of an application, as demonstrated in [14]. This analysis can be carried out without access to a physical GPU. However, static analysis of the assembly code can be tedious and inaccurate, while a full emulation is very slow [15]. The authors of [16] estimate the values of PCs for a given workload from PTX code using a hybrid approach of static analysis and partial emulation where it fails. However, due to the complex nature of GPU native applications and the differences between the PTX instructions and the actual byte code executed by the GPU, the resulting estimates differed significantly from the reference measurements. Additionally, the PTX format is specific to NVIDIA GPUs, rendering this approach infeasible for GPUs of other manufacturers.

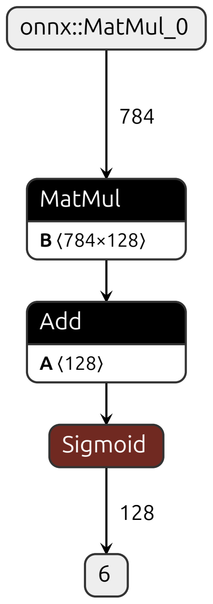


Figure 1: A small, linear ONNX graph combining the MatMul, Add, and Sigmoid operands to form a layer of a deep neural network.

Most of these studies demonstrate their methods on only one [13, 14, 15] or two [8, 11] GPUs, all of them from the same manufacturer.

3 Methodology

In the following section, we describe our methodology. First, we focus on the usage of a representation for ML-models that is manufacturer-independent. Second, we discuss variable selection.

3.1 ONNX

One important drawback shown by the analyzed related work was the dependence of the analysis frameworks on the hardware manufacturers, e.g., the PTX format for NVIDIA GPUs. To have a prediction model that allows taking the hardware architecture as independent variable, we want to focus on a manufacturer-independent approach. A straightforward possibility is the usage of the ONNX format [17]. The ONNX format is an open source community project supported by many companies in the artificial intelligence community, among them hardware manufacturers like AMD, ARM, Intel, and NVIDIA. It defines an extensible computation graph model for machine learning appli-

Model	Category	Architecture	TDP
Jetson AGX Orin	Embedded	Ampere	40 W
GTX 1080	Consumer	Pascal	180 W
Tesla V100S	Datacenter	Volta	250 W

Table 1: Systems under test

cations based on a common set of basic operators and standardized data types. A simple example of such a graph is visualized in **Figure 1**. The graphs are portable across frameworks, can be exported from popular libraries like PyTorch and TensorFlow, and then reloaded by other frameworks.

The ONNX Runtime Project is an open-source project providing hardware-specific implementations of the ONNX operations for different programming models like CUDA, OpenVINO, ROCm, and oneDNN [18]. These implementations, called Execution Providers (EP), enable developers to run the same model on an extensive array of diverse hardware.

Compared to NVIDIA's PTX format, ONNX represents a higher abstraction, focusing on the semantics of the operations while leaving the technical implementation to the EPs. Switching the modelling approach to the ONNX format allows for higher flexibility and portability and allows the approach to be compatible with all platforms supported by an ONNX EP.

3.2 Variable Selection

While GPU Performance Counters have been considered the leading method for predicting energy consumption, [9] found that the counters themselves are specific not only to hardware vendors, but also to hardware architecture. Thus, hardware-agnostic energy models based on these counters are very challenging to design and usually require feature selection specific to each platform [11]. Our approach uses only the semantics of the workload itself in the form of an ONNX file as input, removing the ties to hardware features entirely and mitigating the need for profiling vendor specific counters at runtime.

To generate benchmarking data (which may serve as training data for prediction models in the future), we only profile the power draw of the GPU over time. All modern GPUs keep track of their power consumption, and vendors provide software tools to extract these values. Since the provided power readings are all in Watts, simply replacing the profiling tool within benchmarking and analysis scripts is easily done, yielding comparable results across platforms.

4 Experimental Setup

To provide a proof-of-concept that data interpolation is not sufficient for estimating power consumption, we tested the **Gemm** [19] operation on three different GPUs. In the following section, we describe the choice of GPUs, the process of generating differently sized ONNX graphs, the technical process for benchmarking, and finally the analytical process of benchmarking our data.

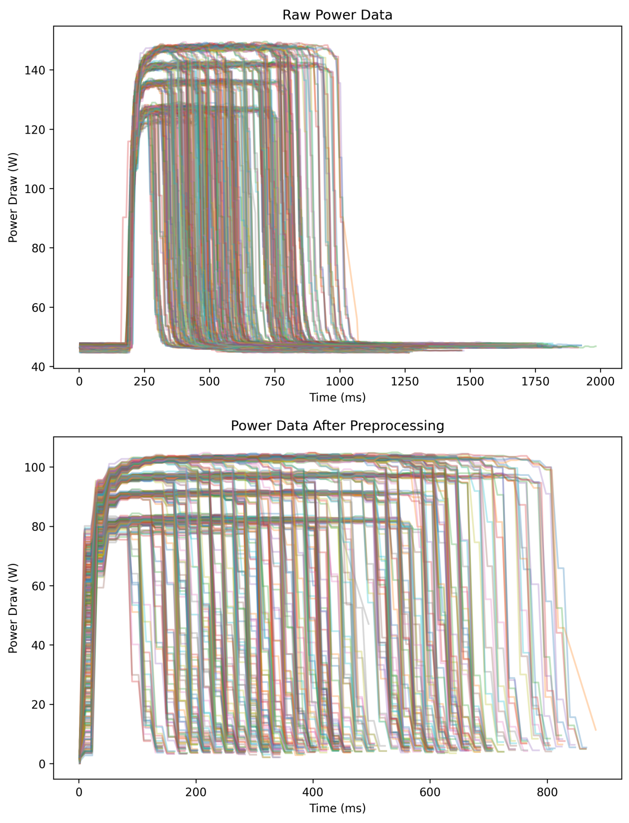


Figure 2: Power data collected on the GTX 1080 before and after preprocessing. The emergent horizontal lines illustrate different maximum power consumption for the different tensor sizes.

4.1 Hardware Selection

The first decision to be made was the choice of GPUs to use for benchmarking. We chose three NVIDIA GPUs that represent three different performance categories and architecture families. First, we chose an older consumer-targeted GPU, the *GTX 1080*. It represents average performance (targeted at consumers) and implements the Pascal architecture. Choices on the lower and upper end of the spectrum were the *Jetson AGX Orin*, a representative for embedded GPUs and the Ampere architecture, and finally the *Tesla V100S*, a datacenter-targeted GPU of the Volta architecture. All three GPUs are listed in **Table 1** along with their category, architecture, and Thermal Design Power (TDP).

For the GTX 1080 and the Tesla V100S, we used the *nvidia-smi* tool to gather power data. Since the Jetson family does not support this tool, we use *tegrastats* to gather data on the Jetson AGX Orin. The operating frequency of the Tesla V100S is fixed to the base frequency to improve the quality of the measurements and the reproducibility of the experiment.

4.2 Graph Generation

To benchmark the energy consumption of the GEMM operation in a controlled way, we need to first define a custom workload with known parameters. We utilize the *onnx* Python package maintained by the open source community to generate computational graphs. These consist of a series of **Gemm** nodes, where the bias input for each node is the result of the previous operation and all other inputs are randomly generated by **RandomUniform** nodes. The random generation of inputs is designed to mitigate potential biases introduced by the choice of specific values and reduce the amount of data moved between the CPU and the GPU. All inputs in the graph have the dimensions $k \times k$, where k is the tensor size referenced in the results below, and consist of 32-bit floating point numbers. Each graph consists of n nodes, where n is a number between 1000 and 5000 chosen randomly for each graph. The range represents a big enough workload to induce meaningful power spikes during computation, and the random choice yields a good distribution to fit a linear regression to in the analysis phase. For a first experiment, we chose $k \in \{500, 550, 600, 650, 700\}$.

The input data for each experiment consists of 25 graphs per tensor size. This number is meant to mitigate artifacts and noise introduced by the generation step. The graphs are created with the above method along with a metadata file containing the filename of the ONNX file, the size of the tensors, the number of nodes, a unique identifier, and the relevant operation for each graph.

4.3 Benchmarking

The benchmarking script reads the metadata file created in the last step to find the filenames for all generated ONNX graphs. One by one, these graphs are then loaded into an ONNX inference session. The EP to be used by the session can be defined via run configuration. Since all three tested GPUs are manufactured by NVIDIA, we use the CUDA EP [20]. The graph is then run once to minimize noise generated by first-execution setup tasks.

After this setup, the profiler is started in a background thread to run alongside the main application. On the GTX 1080 and the Tesla V100S, we use *nvidia-smi* and log the fields *timestamp*, *power.draw*, and *clocks.sm*. As the Jetson platform does not support *nvidia-smi*, we use *tegrastats* instead on the Jetson AGX Orin. Both tools log a timestamp with millisecond accuracy along with the queried data, which is later used to calculate the runtime of the sessions and the total energy consumption over the run. Inference is then run once on the loaded ONNX graph and upon finishing the profiler thread is stopped. The tools write their data to a `.csv`-file named after the unique identifier assigned to the graph at generation.

The sequence of profiler start, model execution, and profiler termination is repeated for a total of five times in order to reduce potential noise introduced during runtime.

4.4 Analysis

In this last step, a script reads the unique identifiers, input sizes, and numbers of nodes for the relevant graphs from the metadata file. For each graph, the `.csv` files corresponding to its five runs are ingested and preprocessed. **Figure 2** shows power data collected on the GTX 1080 GPU before and after the preprocessing step. Each line corresponds to one graph execution and shows the power draw over time during profiling. Before and after the computation, the program sleeps for a while to clearly separate each run from the others and to get a baseline for the static power consumption of the GPU. The plot shows all data collected for the GEMM operation on the GTX 1080, including all five input sizes. There are distinct horizontal lines visible in the data, where the different graphs reach their maximum in terms of power draw, which stem from the different levels of GPU utilization for different tensor sizes.

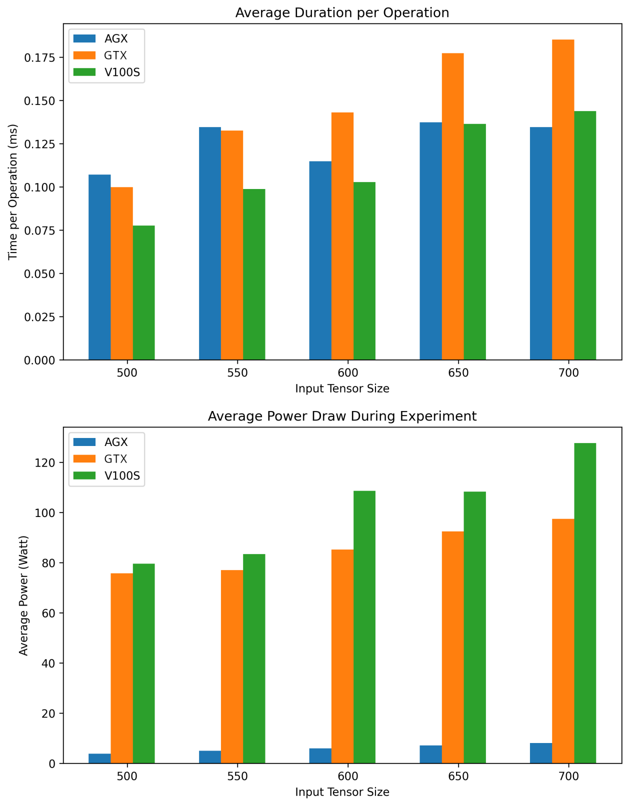
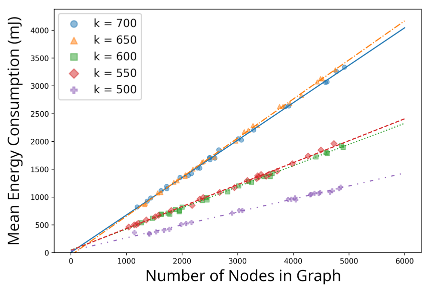
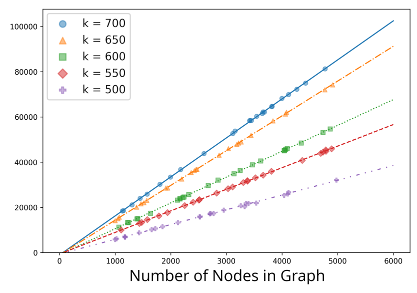


Figure 3: Average compute time per operation and average dynamic power consumption during execution for different input tensor sizes k.

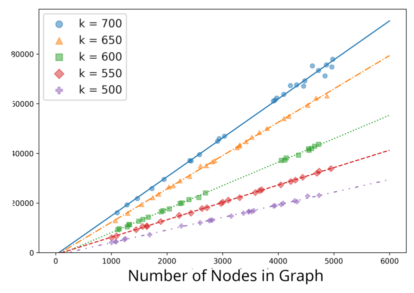
During preprocessing, the timeframe of actual activity is determined by examining the rate of change of the power data. The sections at the start and end of the data window are considered idle time and not part of the dynamic power consumption. The threshold can be adjusted to fit different power profiles and profiling tools. This step also aligns the data series, revealing a resolution of around 20 ms for the measured power. The minimum power consumption during this timeframe is interpreted as the idle power of the GPU and subtracted from the measured values to retrieve the dynamic power during the operation. The area under the resulting graph is then computed as the energy consumption of the graph in question. The energy consumption of all five runs for a given graph is averaged. The results for all graphs corresponding to a given input size are then plotted against the number of nodes in the respective graphs and a linear regression is fitted to the data using the Ordinary Least Squares solver of the `statsmodels` library [21].



(a) Results for the Jetson AGX Orin.



(b) Results for the GTX 1080.



(c) Results for the Tesla V100S.

Figure 4: Benchmarking results for all three GPUs on tensor sizes $k \in \{500, 550, 600, 650, 700\}$ with varying node counts per graph between 1000 and 5000. Points show averaged observations, colors represent different tensor sizes. Linear regression trends are shown by lines of the respective color.

5 Results

This section describes the results of our benchmarking experiment. In the beginning, we discuss universal results of our analysis, then we describe the results of the trend analysis for all tensor sizes and GPUs. Afterwards, we analyze the resulting average power consumption per operation, which provides the first validation of our hypothesis. Finally, we strengthen our results by rerunning several benchmarking setups with a more fine-grained analysis.

5.1 Universal Results

First, **Figure 3** demonstrates the average duration per operation as well as the average power draw during the experiment. The Jetson AGX Orin, which has by far the lowest TDP, has the lowest average power consumption by a wide margin. While the average power draw of the GTX 1080 and the V100S are closer for smaller tensor sizes than for larger ones, the consumption of the V100S is noticeably higher overall. This increased power consumption is accompanied by a better performance in terms of processing speed, as the V100S shows distinctly better runtimes than GTX 1080. The Jetson AGX Orin, despite its low power draw, is comparable in runtime to the GTX 1080 for the smaller tensor sizes, and even to the V100S for larger ones. While most of the results confirm expected behavior, the Jetson AGX Orin's performance requires further analysis.

5.2 Analyzing Trends for Different Tensor Sizes

Besides the general benchmark results, we also analyzed the power consumption in more detail. **Figure 4(a) to 4(c)** shows total energy consumption over a single graph execution on the y-axis as determined by the number of nodes in the graph (x-axis) and the input tensor size (color of dots). For every tensor size setup, we fitted a regression line to the data, which can be seen as trend lines in the figures. All trend lines show a linear behavior for increasing node count. With increasing input tensor size ($k \in \{500, 550, 600, 650, 700\}$), the slopes generally become steeper. The slopes of these linear regressions will be interpreted as the estimated energy consumptions per

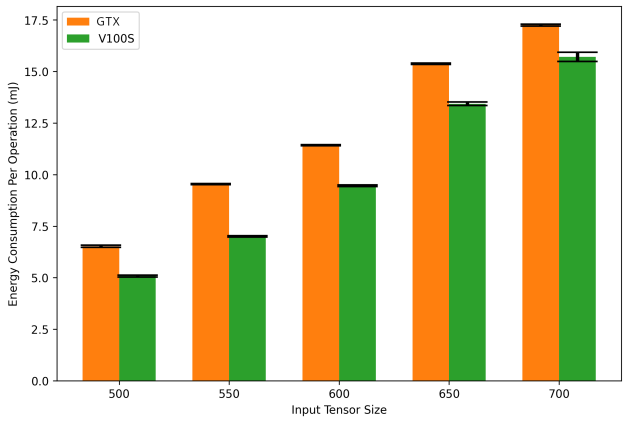


Figure 5: Total energy consumption per operation for different input sizes on the GTX 1080 and Tesla V100S.

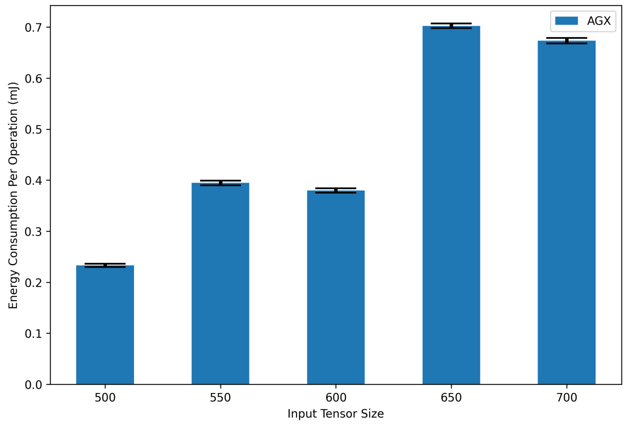
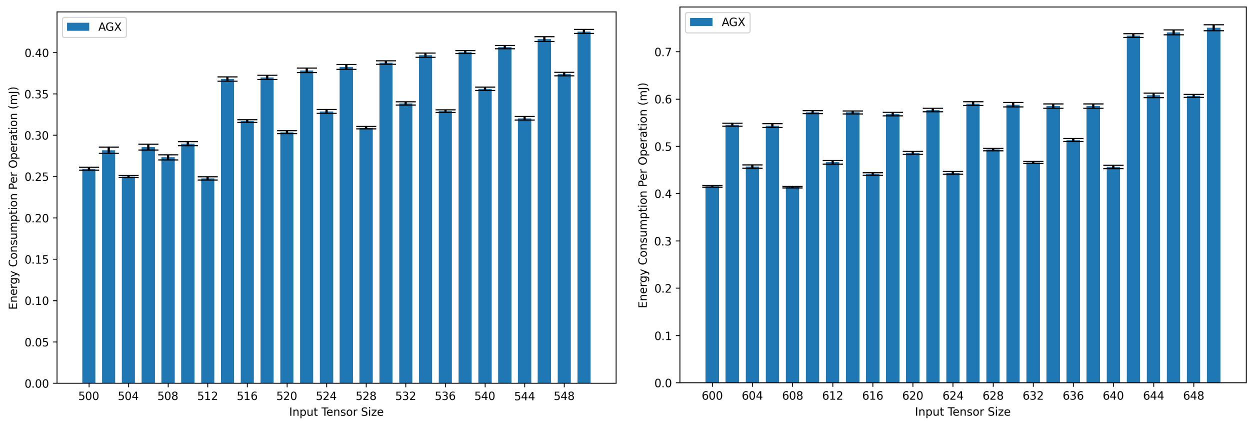


Figure 6: Total energy consumption per operation for different input sizes on the Jetson AGX Orin.



(a) Results for k between 500 and 550. (b) Results for k between 600 and 650.

Figure 7: Total energy consumption per operation for more-fine grained tensor sizes on the Jetson AGX Orin.

GEMM operation for the respective tensor size k.

Comparing the trend lines, the three GPUs show clear differences: While the lines for the V100S appear evenly spaced (especially for the three lower tensor sizes), the GTX 1080 shows a tendency towards having closer trend lines for $k = 550$ and $k = 600$, as well as for $k = 650$ and $k = 700$. This pattern is put to an extreme for the Jetson AGX Orin, where $k = 550$ and $k = 600$ are very close to one another and even switch positions, as well as $k = 650$ and $k = 700$. Additionally, for the Jetson AGX Orin, the linear trends are not as clear as for the other GPUs.

Figures 5 and 6 show the estimated slopes from the trend lines in Figure 4 along with error bars showcasing the standard error for the regression as reported by `statsmodels`. The clarity of the linear trends in Figure 4(b) and 4(c) is shown in the very small error bars in Figure 5. The same holds for the depiction of the trend lines and estimated slopes for the Jetson AGX Orin in Figures 4(a) and 6. The linear regressions were able to accurately approximate the slopes underlying the data with a maximum standard error percentage of 1.42%.

Figure 5 demonstrates the estimated additional energy consumption per additional node in the computational graph for different input tensor sizes. It can clearly be seen, that the GTX 1080 shows higher values than the V100S, indicating a higher additional power consumption per operation. One can also see the bigger increase in slope estimation for the GTX 1080 from $k = 600$ to $k = 650$.

Figure 6, however, shows this pattern much more clearly, as the bars for $k = 600$ and $k = 700$ are lower than those for $k = 550$ and $k = 650$, respectively. Although the differences here are not very big, the pattern does clearly deviate from the proportionally increasing behavior of the GTX 1080 and the V100S demonstrated in Figure 5. This behavior was very surprising and motivated a more fine-grained analysis with $k \in \{500, 502, 504, \ldots, 550\}$ and $k \in \{600, 602, 604, \ldots, 650\}$.

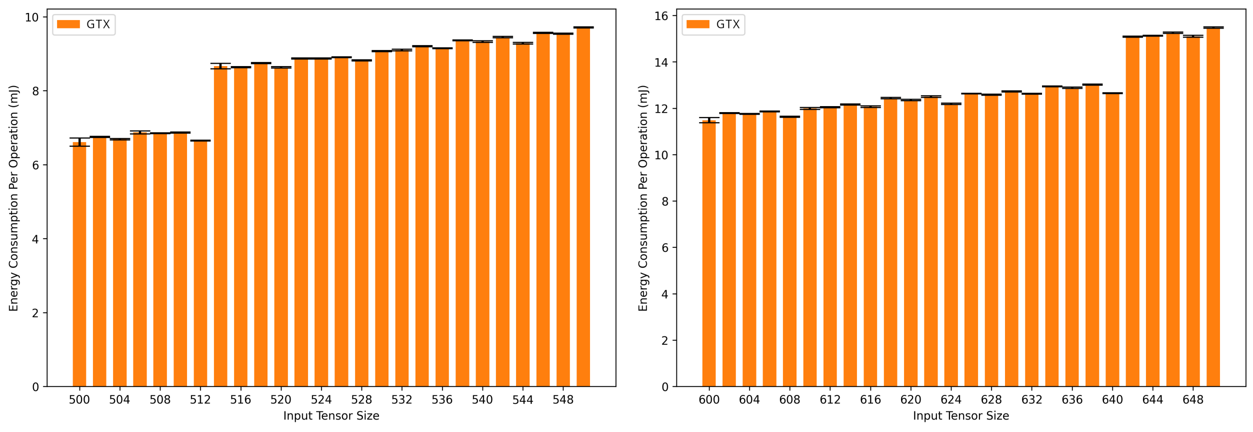
5.3 A More Fine-Grained Analysis of Tensor Size Influence

The results of our more fine-grained analysis of the influence of input tensor size on the energy consumption, as measured on the Jetson AGX Orin, are shown in **Figure 7**. Figure 7(a) shows the results for the smaller tensors, while Figure 7(b) shows the results for the larger tensor sizes. There are several things worth noting in these results:

They show a prominent and sudden increase in energy consumed per operation for k values above 512. This pattern can also be seen for the larger tensors with sizes larger than 640.

Further inspection reveals three different trends: First, every second tensor size has a higher energy consumption per operation than the others, with a linear trend emerging within these tensor sizes. Of the tensor sizes with lower energy consumption, every second one displays an even lower value, while both groups still show roughly linear trends. The tensor sizes with higher energy consumption are the ones *not* divisible by four. For the tensor sizes divisible by four, a distinctly lower energy consumption can be noticed, which is further decreased for input sizes divisible by 8. The energy consumption at $k = 544$ appears to be even lower than suggested by the observed pattern, which may be due to the fact that 544 is divisible by 32.

Figure 8 shows data collected in the same way on the GTX 1080. This data also shows sudden increases in energy consumption for input tensor sizes larger than 412 and 640. For some input sizes which are divisible by multiples of 4, like 512, 520, 624, and 640, we can also observe a lower energy consumption per operation compared to their neighbors. However, this pattern is not nearly as pronounced as for the Jetson AGX Orin. This result is consistent with our findings from Section 5.2, where we observed trend lines for the GTX 1080 which were not quite evenly spaced, but much more evenly than those of the Jetson AGX Orin.



(a) Results for k between 500 and 550.

(b) Results for k between 600 and 650.

Figure 8: Total energy consumption per operation for more-fine grained tensor sizes on the GTX 1080.

6 Discussion

The results of our experiments show differences in the hardware platforms under test, not only in overall efficiency, but also in patterns. While both the Jetson AGX Orin and the GTX 1080 appear to show a sudden increase of power consumption per operation with respect to input size in intervals of 128, the Jetson additionally shows a great increase in efficiency for input sizes divisible by 4 or 8.

The Jetson AGX Orin is the tested GPU with the most modern architecture, enabling it to make use of NVIDIA's tensor compute cores and custom 32-bit Tensor Float format (TF32). The CUDA EP only activates this feature for GPUs with the Ampere architecture or later [20], as earlier GPUs do not support the TF32 format [22]. According to the NVIDIA documentation, tensor cores can provide a great speedup for matrix multiplication with the most efficiency provided if the matrix dimensions are divisible by 4 [23]. This explains the pattern seen for the AGX Orin as well as its absence from the older GTX 1080.

The results show, that energy consumption is not always correlated with workload parameters, like input size, in a straightforward way, and that the actual relationship between these quantities is hardware specific. Consequently, models aiming to estimate energy consumption cannot rely on the extrapolation of simple models or insights gained from a single hardware architecture, but need to learn these relationships in order to produce accurate results, highlighting the need for a reliable and portable method to synthesize training data from diverse hardware platforms.

Although the choice of the ONNX framework allows our approach to be generalized to most ML frameworks and hardware platforms, the high abstraction level of the format can obscure details in the implementation, especially when multiple EPs are considered, and thus hinder targeted reasoning about these relationships. However, given clean and complete training data, a fitting ML method like neural networks or random forests may be able to learn the hidden patterns and generate accurate predictions.

While ONNX supports a large variety of hardware platforms and is extensible, using it for platforms without official or mature support for EPs is not feasible, limiting the total number of platforms eligible for analysis. For example, there is no EP for Hailo accelerators, which can be combined with a Raspberry Pi for a low-cost edge device.

Additionally, the benchmarking and execution is reliant on the implementation of the EPs. For Ampere GPUs or later, the CUDA EP also automatically enables the use of the TF32 format, which can provide significant speedups. While research by NVIDIA shows the same convergence-to-accuracy behavior for models using TF32 as for those using standard floating point numbers [22], TF32 does sacrifice some precision in the form of mantissa bits, holding potential for divergent model performance.

Threats to Validity of this work include the choice of thresholds used for the preprocessing of measurement data as described in Section 4.4, which was chosen by hand for each platform. Care has been taken to select proper values, and the subtraction of the static power consumption drastically decreases the impact of the size of chosen windows of activity on the calculation of overall energy consumption. However, the displayed computation times could be impacted by suboptimal threshold choices.

An additional threat could be posed by differing driver versions on the systems used for testing. Due to limitations in the systems used for testing, slightly different versions of the CUDA and cuBLAS libraries were used. Although GEMM is a standard operation and behavior should not vary too much across library versions, these differences could possibly impact the exact computations carried out during benchmarking.

Although the GEMM operation is a highly prominent and relevant operation in ML, it does not capture the whole behavior of real-world ML workloads. Effects like Operator Fusion cannot be characterized through this synthetic workload.

While the GTX 1080 and the V100S are discrete GPUs connected via a PCIe interface, the AGX Orin is designed as a System-on-a Chip. This difference affects the behavior of the systems regarding data movement and kernel startup costs, as the PCIe interface displays significantly higher latency than custom designed connections on a chip. Although the setup was designed to reduce data movements between CPU and GPU, this difference may have a significant impact on the measured runtimes and power consumption.

7 Conclusion and Future Work

In this paper, we presented a hardware-agnostic approach for benchmarking the energy consumption of ML workloads using the ONNX format and ecosystem. We showed a first proof-of-concept, in which we benchmarked the consumption of the GEMM operation, illustrating the ability of our approach to reveal hardware specific patterns in energy efficiency, as well as its portability across different types of GPUs with different architectures. We also showed that patterns in energy consumption can vary significantly on different hardware platforms, highlighting the need for data-based estimation models and portable benchmarking methods.

In the future, we plan to extend our experimentation to other operations within the ONNX framework to show the versatility of the approach, and validate its portability to GPUs made by other manufacturers. We also plan to extend the scope of experimentation from a single type of operation to more complex graphs including multiple operation types.

Acknowledgment

This work was supported by the Data Science Center of the University of Bremen, funded by the State of Bremen.

The authors would like to thank Christina Plump for many helpful discussions.

References

[1] C. Rahal, M. Verhagen, and D. Kirk, "The rise of machine learning in the academic social sciences," *AI & SOCIETY*, vol. 39, pp. 799–801, Aug. 2022.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[3] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, Apr. 1980.

[4] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1, pp. 21–28, 1988.

[5] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, IEEE, Aug. 2018.

[6] I. Penn and K. Weise, "Hungry for energy, amazon, google and microsoft turn to nuclear power," *The New York Times*, Oct. 2024.

[7] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU power: A survey of profiling, modeling, and simulation methods," *ACM Computing Surveys*, vol. 49, pp. 1–27, Sept. 2016.

[8] N. Bombieri, F. Busato, F. Fummi, and M. Scala, "Mipp: A microbenchmark suite for performance, power, and energy consumption characterization of GPU architectures," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, IEEE, May 2016.

[9] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Energy predictive models of computing: Theory, practical implications and experimental analysis on multicore processors," *IEEE Access*, vol. 9, pp. 63149–63172, 2021.

[10] C. A. Metz, M. Goli, and R. Drechsler, "ML-based power estimation of convolutional neural networks on GPGPUs," in *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pp. 166–171, IEEE, Apr. 2022.

[11] A. Alnori, K. Djemame, and Y. Alsenani, "Agnostic energy consumption models for heterogeneous GPUs in cloud computing," *Applied Sciences*, vol. 14, p. 2385, Mar. 2024.

[12] C. Rodriguez, L. Degioanni, L. Kameni, R. Vidal, and G. Neglia, "Evaluating the energy consumption of machine learning: Systematic literature review and experiments," 2024. arXiv:2408.15128 [cs.LG].

[13] P. Delestrac, J. Miquel, D. Bhattacharjee, D. Moolchandani, F. Catthoor, L. Torres, and D. Novo, "Analyzing GPU energy consumption in data movement and storage," in *2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 143–151, IEEE, July 2024.

[14] C. A. Metz, M. Goli, and R. Drechsler, "Towards neural hardware search: Power estimation of CNNs for GPGPUs with dynamic frequency scaling," in *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, MLCAD '22, pp. 103–109, ACM, Sept. 2022.

[15] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IEEE, May 2013.

[16] C. A. Metz, C. Plump, B. J. Berger, and R. Drechsler, "Hybrid PTX analysis for GPU accelerated CNN inferencing aiding computer architecture design," in *2023 Forum on Specification & Design Languages (FDL)*, pp. 1–8, IEEE, Sept. 2023.

[17] ONNX Community, “Onnx: Open neural network exchange.” https://onnx.ai, Jan. 2026.

[18] ONNX Runtime Community, “Onnx runtime: Accelerated cross-platform machine learning.” https://onnxruntime.ai/, Jan. 2026.

[19] ONNX Community, “Onnx 1.21.0 documentation: Gemm.” https://onnx.ai/onnx/operators/onnx__Gemm.html, Jan. 2026.

[20] ONNX Runtime Community, “Cuda execution provider.” https://onnxruntime.ai/docs/execution-providers/CUDA-ExecutionProvider.html, Jan. 2026.

[21] Statsmodels Community, “statsmodels.regression.linear_model.ols.” https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html, Dec. 2025.

[22] NVIDIA Corporation, “Tensorfloat-32 in the A100 GPU accelerates AI training, HPC up to 20x.” https://blogs.nvidia.com/blog/tensorfloat-32-precision-format/, May 2020.

[23] NVIDIA Corporation, “Matrix multiplication background user’s guide.” https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html, Jan. 2026.