Analyzing Energy and Performance Trade-offs for Network Anomaly Detection based on Deep Learning

Tiago Torres Schmidt[1], **Lisandro Z. Granville**[1], **Alberto Schaeffer-Filho**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{ttschmidt, granville, alberto}@inf.ufrgs.br

Abstract. *Network anomaly detection based on Deep Learning has already achieved outstanding performance results. However, the performance obtained by deep learning solutions is partially explained by the large scale of such underlying models. This paper studies the energy and performance trade-offs for deep learning models and their hyperparameter configurations when applied to network anomaly detection. The paper proposes an energy and performance profiling mechanism to observe the results obtained from different configurations of a given model, using a combination of statistical and instrumented profiling.*

Resumo. *A detecção de anomalias em redes baseada em aprendizado profundo já atingiu resultados impressionantes. No entanto, a performance obtida por modelos de aprendizado profundo é parcialmente explicada pela grande escala de tais modelos basilares. Este artigo estuda as compensações entre energia e performance para modelos de aprendizado profundo e suas configurações de hiperparâmetros, quando aplicados em detecção de anomalias em redes. O artigo propõe um mecanismo de perfilação de energia e performance para observar os resultados obtidos de cada configuração diferente de um determinado modelo, usando uma combinação de perfilação estatística e instrumentada.*

1. Introduction

According to Forbes [Ene 2023], the cost of cyber-crime is projected to hit an annual USD 10.5 trillion by 2025. In search of a solution to security problems, machine learning (ML) and deep learning (DL) have been widely used and explored for the task [Wang et al. 2021]. For example, Vikram and Mohana [Vikram and Mohana 2020] use an unsupervised model to provide anomaly detection with an AUC score of 99.9%. This kind of accuracy and modeling for anomaly detection has been reproduced many times. For example, Kostas [Kostas 2018] discusses, proposes, and tests seven different ML and DL models for anomaly detection. In all models, great performance results were achieved, with F-measures of 0.99 during the testing cycle.

Although great performance standards have already been achieved, machine learning solutions demand significant computing power, especially using complex models such as deep neural networks [Bianco et al. 2020]. The computing resource cost applied in training and inference for these DL network anomaly detection solutions contradicts sustainable and renewable energy trends. As estimated previously, a neural network model can generate approximately five times the average carbon dioxide emissions in the lifetime

of a car [Candelieri et al. 2021]. Fortunately, the evolution of ML and DL has already led to research efforts on green ML [Saeed et al. 2022] and green artificial intelligence [Yang et al. 2020].

This paper proposes a comparative analysis of performance and energy for deep learning-based network anomaly detection systems, using instrumentation profiling based on performance counters combined with an exploration of hyperparameters of the underlying model. The objective is to explore the trade-offs between performance and energy consumption, detailing the intrinsic relationship of a hyperparameter adjustment with its respective gain or loss in performance or energy consumption. The profiling approach used is a combination of statistical profiling and instrumentation profiling, utilizing Hardware Performance Counters (HPC) to extract the energy consumption and also deep learning libraries for the performance metrics. The different hyperparameter configurations profiled are provided by a grid search algorithm that explores a pre-defined range of values. The paper's main contributions are: *(i)* combining hardware performance counters with specialized deep learning software to provide precise energy and performance metrics; *(ii)* exploring energy and performance trade-offs of different hyperparameter configurations for deep learning, utilizing a grid search algorithm; and *(iii)* providing a comparative analysis for various deep learning models in terms of energy and performance.

The remainder of this paper is organized as follows. Section 2 discusses the theoretical background required for this paper. Section 3 presents an overview of the related work. Section 4 describes the framework proposed to evaluate the trade-offs between energy consumption and performance. Section 5 discusses the prototype and experimental evaluation. Finally, Section 6 presents the concluding remarks and future work.

2. Background

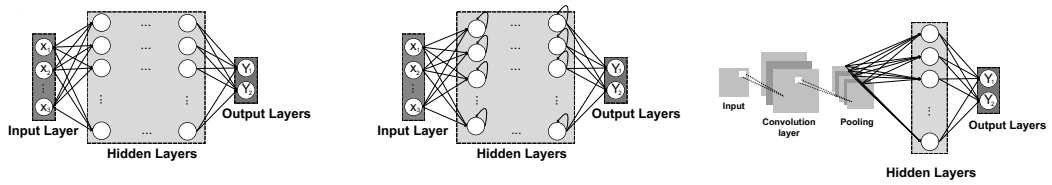
This section presents a brief overview of network anomaly detection based on DL and energy consumption measurement.

2.1. Deep Learning for Network Anomaly Detection

Deep Learning is a subset of Machine Learning, where all models present a representation learning stage (also called feature learning stage) and multiple layers [LeCun et al. 2015]. More recently, ML and DL for network anomaly detection have received significant attention [Boutaba et al. 2018]. This is mainly due to the autonomy and robustness offered. ML and DL techniques tend to extract the subtle pattern of the general anomaly (*e.g.*, a network attack), which means that these models have the capability of detecting previously unknown anomalies [Marnerides et al. 2014, Santos da Silva et al. 2016].

Some of the predominant DL models for network anomaly detection in the literature are Multi-Layer Perceptron (MLP), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN) [Boutaba et al. 2018]. These are briefly discussed below:

Multi-layer Perceptron is a machine learning technique based on a feed-forward deep network, in which each layer is a collection of independent *perceptron's*, with all units of a layer connected to all units of the next layer, forming the feed-forward condition. Data is passed from the first layer (input layer), which propagates to the following layer, and so on [Teoh et al. 2018]. After the input is propagated throughout the network, a



(a) MLP model (b) LSTM model (c) CNN model

Figure 1. Deep Learning Models.

final result is generated and compared to an answer with an error function. Then, a back-propagation system will adjust all *perceptron's* weights and biases aiming for a better answer in the next iteration (Figure 1(a)).

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network. An RNN is an MLP with cyclic connections on each unit [Kim et al. 2016]. Those cyclic connections enable the network to mimic the memory process of humans. The major limitation of simple RNNs is their difficulty in training due to vanishing or exploding gradients [Bengio et al. 1994]. To address this issue, Hochreiter and Schmidhuber introduced the LSTM architecture [Hochreiter and Schmidhuber 1997]. Unlike traditional RNNs, LSTMs include a forget gate that controls if the previous state is remembered. This mechanism helps prevent gradients from vanishing or exploding, allowing LSTMs to learn long-term dependencies more effectively (Figure 1(b)).

Convolutional Neural Network is a DL architecture designed for handling high-dimensional data with spatial correlation. The main component of a CNN is the *convolutional* layer. A convolutional layer takes the *input* and *convolves* it with a set of filters, called *kernels*, to produce an output. Mathematically, the convolution process is performed by sliding the set of filters over the input and applying the dot product between input and filter [Naseer et al. 2018]. The secondary main component of a CNN is the *pooling* layer, designed to control the overfitting. The pooling layers act by decreasing the size of the input representation with a sampling technique. These are combined in a Deep Neural Network structure to generate CNN architectures (Figure 1(c)).

2.2. Energy Consumption Measurement

There is a diverse range of methodologies to quantify energy consumption [Goel et al. 2012]. Traditional approaches, such as outlet watt meters, precisely measure the total energy consumption of a computer but are often inaccessible. Alternatively, software estimation correlates energy consumption with the frequency of function or module calls, representing the primary contributors to a program's energy usage. While this approach is much more accessible, its results are often unclear and imprecise.

A precise and easily accessible methodology is to measure energy consumption using hardware performance counters. HPC are special purpose registers that measure micro-architectural events and record values at native execution speed [Demme and Sethumadhavan 2011]. Thus, they offer an excellent compromise between accessibility and precision. However, to access the information stored in those registers at the user level, a vendor-specific interface is required. The most established interfaces for

HPC are the Intel Running Average Power Limit (RAPL) and the NVIDIA Management Library (NVML).

Running Average Power Limit is a platform-specific power management interface for the Intel Sandy Bridge architecture and onwards. Inside RAPL-compatible chips, the internal circuitry is responsible for registering the HPC information on Model Specific Registers (MSRs) [Weaver et al. 2012]. For example, access to those registers in Linux is done via a directory called $/dev/cpu/N/msr$ (where N is the number of the core), and it requires ring-0 permission. However, the Linux "MSR driver" enables read-only access to the file, without the ring-0 permission.

RAPL interface exposes *power domains*, each representing the energy consumption of a section. The **Package** N *domain* measures the consumption of the N entire processor socket (including all *Powerplanes*, last-level cache, and the memory controller). The **Powerplane** 0 *domain* measures consumption of all cores, while **Powerplane** 1 measures consumption of the embedded GPU (if available). The **DRAM** *domain* measures the energy consumption of all DRAM DIMM's. Lastly, the **Psys** *domain* measures the consumption of all systems on the chip, including the Platform HUB Controller (PCH) and even peripheral components (embedded DRAM - eDRAM - for bus management).

NVIDIA Management Library is a C-based API for monitoring NVIDIA GPU devices. It allows developers to query GPU device states, such as utilization, clock rates, etc. More importantly, it provides access to the power draw by reading the onboard sensors. NVML has been available since CUDA V4.1, released in 2011. Also, NVIDIA has the NVIDIA System Management Interface (nvidia-smi), a dedicated command line application. It is designed to be a utility tool, built upon the NVML API [Arafa et al. 2020].

3. Related Work

This section discusses studies on hyperparameter tuning oriented by performance and resource consumption, energy-constrained DL models for network anomaly detection, and general ML model energy analysis and modeling. These are summarized in Table 1.

Preuveneers et al. [Preuveneers et al. 2020] propose a two-stage framework for ML model hyperparameter tuning that considers resource usage. The first stage generates different possible configurations of hyperparameters that obtain similar F1 scores. The second stage simulated a test environment, while the resource usage was recorded. In the paper, the authors considered only one ML model each time; the F1 score was the sole performance metric considered, and the resources monitored were execution memory usage, execution elapsed time, and model size (in memory).

Hsu et al. [Hsu et al. 2018] propose a framework for multi-objective hyperparameter optimization of CNNs. The demonstration complemented accuracy with energy consumption optimization. The first stage employs an RNN with a reward system to generate different configurations of hyperparameters. The second stage simulates the training process of a CNN, measuring the desired metrics. The work considered two different types of CNN models in the experiments (AlexNet and CondenseNet), and the metrics observed were accuracy, energy (per 1,000 inferences), and peak power (in Watts).

Sedjelmaci et al. [Sedjelmaci et al. 2016] propose an anomaly detection approach, specialized for low-resource Internet of Things devices. The authors combined game the-

Table 1. Related Work on Energy-Aware Machine Learning Model Development.

Reference	ML techniques	Metrics	Description
Preuveneers et al. 2020	Support Vector Machine, Random Forest.	Memory (MB), Wall Clock Time (ms), Object Size (MB), CPU time (ms).	Hyperparameter tuning using resource usage and F1-score.
Hsu et al. 2018	AlexNet, CondenseNet, ResNet.	Accuracy, Energy (Joule), Peak Power (Watts).	Hyperparameter tuning using accuracy and another user-selected metric.
Sedjelmaci et al. 2016	MLP	Accuracy, and node general energy consumption.	A tailored model for low-resource Internet of Things Networks
Tripp et al. 2024	MLP in different DL setups.	Loss, FLOPS, and energy consumption (with great precision).	In-depth analysis of energy consumption, loss, and FLOPS relationships.
Rodrigues et al. 2024	AlexNet, GoogleNet, and others	Non-energy related PMCs and average execution time.	Energy measurement and modeling for CNN in a specific embedded platform.

ory, signature detection, and anomaly detection. The authors compiled several existing models from the literature to benchmark and compare their novel approach. The benchmark simulated and recorded metrics using sensor networks simulation software. Evaluation metrics included accuracy and per-node energy consumption across the network.

Tripp et al. [Tripp et al. 2024] executed an empirical benchmark of the energy consumption and efficiency of deep neural networks. To measure the energy consumption, node-level Watt-meters were used. The authors extracted measurements for many different deep neural network configurations available in a DNN configuration dataset. The energy consumption results were compiled against the respective hyperparameter configurations, enabling the authors to analyze and model energy consumption.

Rodrigues et al. [Rodrigues et al. 2018] propose a framework for *fine-grained* performance and energy measurement and prediction of CNN models in an embedded platform. Their framework results from the integration of vendor-specific tools and a DL framework. The testbed used for the framework was a group of different CNN models, including AlexNet, GoogleNet, and others. The initial results derived from the framework measurement enabled the creation of an energy prediction model based on processing-related HPC (but not using energy-related HPC).

Most of the papers discussed above ignore or use imprecise metrics for energy consumption, or are focused only on the comparison or tuning of a single ML model. That is, most of the works do not use a precise method as HPC to measure energy (some do not even consider energy consumption), and only choose one specific DL model at a time (picking either MLP or CNN only). Differently, our approach aims to analyze different DL models, varying their hyperparameter configurations in terms of performance and energy consumption. The energy consumption methodology that we present is based on precise measurements obtained from HPCs, and the set of performance metrics that we discuss is more extensive than in the related work.

4. Energy Profiling Framework

This section describes the framework we propose to analyze the trade-offs between energy consumption and performance for network anomaly detection based on DL. The following subsections describe details about the profiling framework.

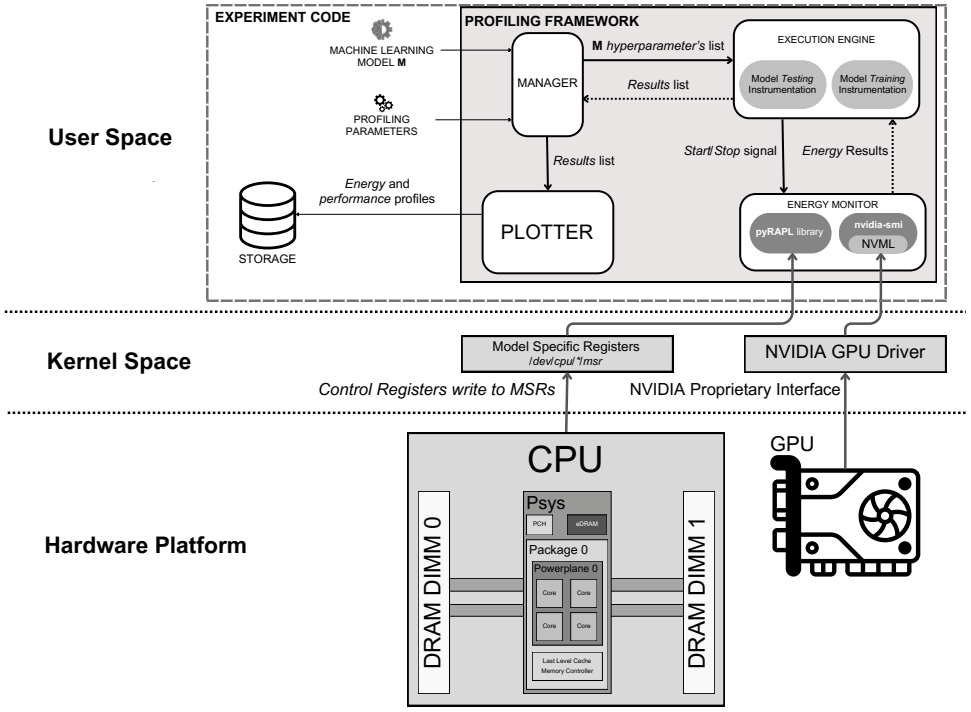


Figure 2. Energy Profiling for Network Anomaly Detection Framework.

4.1. Framework Overview

The fundamental objective of the profiling framework is to facilitate the analysis of relationships between energy consumption, performance, and DL model hyperparameter configurations. The primary input is a user-defined DL model for network anomaly detection. The framework outputs a collection of energy and performance line plots for each performance metric, hyperparameter, and different model configurations. The framework focuses on the understanding of energy and performance trade-offs, thus a simple grid search approach was selected to explore the hyperparameter configurations. This naive yet extensive exploration is sufficient to cover the set of possible values for each hyperparameter.

The proposed framework comprises a set of key modules: an `EnergyMonitor`; an `ExecutionEngine`; a `Plotter`, and a `Manager`[1]. In Figure 2, we show an overview of the modules that comprise the framework and their interactions. In particular, the **Intel RAPL** energy consumption collection stack is illustrated. In summary, the chip's internal circuitry redirects the information from the HPC to control registers. Then, the control registers write to the MSRs via a kernel driver.

After being available in the MSRs, our framework utilizes the pyRAPL library to collect the MSR information, organized in *domains*. As our analysis is focused on the energy cost of inference and training for models, the most precise measurement for energy is from the *Package 0 domain*, as it represents the energy cost of all cores, cache structures, and memory controller. The pyRAPL implementation exposes the total consumed energy, during the execution of a code block (between the class `pyRAPL.Measurement`

[1]In this paper, due to space restrictions, the Plotter and Manager modules will not be described. The Manager is summarized as a framework coordination unit.

`.begin()` and `.end()` methods). The total energy is represented in micro Joules (mJ).

The **NVIDIA NVML** and **nvidia-smi** energy consumption collection stack is also displayed. Unfortunately, the internal details of the NVIDIA stack are not available. However, the GPU energy consumption information is propagated initially by the NVIDIA driver (kernel space). Then, the NVML library (user space) reads the driver energy consumption information, making it available through the *nvidia-smi* command line tool. The NVIDIA stack only reads the current power consumption of the GPU, expressed in Watts.

To describe the novelty of the proposed approach, the `ExecutionEngine` and `EnergyMonitor` modules will be briefly characterized below.

4.2. Execution Engine

The `ExecutionEngine` is instantiated by the `Manager`, receiving a list of all hyperparameter configurations that will be profiled, along with environmental parameters, and a list of performance metrics requested by the user (*e.g.*, 'accuracy', 'precision', 'recall', 'F1-score'). For each hyperparameter configuration, the `ExecutionEngine` configures the environmental conditions requested, such as the computing platform, the batch size, and the sampling rate. Then, it prepares and configures the user-provided base DL model with the current hyperparameter configuration. Once the model is prepared, it signals the `EnergyMonitor` to start recording the energy consumption values. The `ExecutionEngine` then executes several training or testing iterations for statistical validation.

Performance Metric Instrumentation (Training): The training process uses the Keras[2] package function `fit()`. The training utilizes 80% of the dataset and proceeds a specified number of epochs. After each epoch, the model's performance is evaluated on the remaining 20% of the dataset. After training is complete, the `ExecutionEngine` evaluates all performance metrics specified by the user and also saves a copy of the model (and weights) to local storage.

Performance Metric Instrumentation (Inference): The `ExecutionEngine` loads the trained model (and weights) from local storage. The model's performance is evaluated on the entire dataset with only one pass per statistical repetition (differently from training, which performs repetitions according to the number of epochs), utilizing the Keras `evaluate()` function. When all statistical repetitions are completed, the `ExecutionEngine` signals the `EnergyMonitor` to stop recording energy consumption, retrieves the recorded energy consumption values, and aggregates them with the runtime results (for performance and elapsed time). The results are summarized by averages and saved to a consolidated results list.

4.3. Energy Monitor

The `EnergyMonitor` is initialized solely with communication interfaces to the `Manager` and the `ExecutionEngine`. It operates passively, awaiting signals from other units to initiate activity. Upon receiving a signal from the `ExecutionEngine` to commence recording energy consumption, the `EnergyMonitor` also receives the target computing platform (CPU or GPU). Subsequently, it initiates the energy consumption recording process.

[2]https://keras.io/

CPU profiling: The `EnergyMonitor` utilizes the Intel RAPL library to obtain CPU energy consumption data. It records the initial energy consumption value at the beginning of the profile and the final value at the conclusion. The difference between these values represents the energy consumed during the number of repeated profiling routines (for statistical validation), leading to the average energy consumed by the routine executed (training or inference).

GPU profiling: The `EnergyMonitor` periodically polls the NVIDIA NVML library to obtain the current power consumption of the GPU. These power readings are collected throughout the profiling period (5 ms). Following the completion of the profiling period, the `EnergyMonitor` calculates the average power consumption and transmits this value to the `ExecutionEngine`.

5. Implementation and Experimental Results

This section will describe the implemented prototype, the experimental setup, and the experimental results.

5.1. Prototype and Experimental Setup

For this paper, a prototype for the framework was developed in Python. The source code is publicly available in GitHub[3]. The Python version used for development was 3.12.3. The full list of Python dependencies is presented in the repository files. Additionally, the repository contains unit tests for the framework modules and examples of usage. The experiments were executed on a 13th Gen Intel i5-13400 CPU (16 cores with a 3.2 GHz clock frequency), L1 Cache of 80 KB (performance cores), L2 Cache of 1.25 MB (performance cores), with 16 GB RAM, and on a GPU NVIDIA GeForce RTX 3060 Ti, L1 Cache (per Stream Processor) of 128 KB, L2 Cache of 4 MB, Base Clock of 1410 MHz, and Boost Clock of 1665 MHz.

The experiment consists of profiling three different DL models, and observing energy consumption, over three different ML performance metrics. On top of that, the training process runs on a GPU, and the testing process on a CPU. The observed metrics for this experiment were precision, F1-score, and recall. Table 2 summarizes the values of hyperparameters considered for this experiment. The dataset used for this experiment is the NSL-KDD dataset [Tavallaee et al. 2009]. The anomalies in the NSL-KDD dataset include denial of service, probing, unauthorized access from remote machines, and unauthorized acquisition of superuser privileges. The total number of dataset entries available is 125,973 [Dhanabal and Shantharajah 2015].

Table 2. Experiment Hyperparameters Values.

Hyperparameter	Values
Number of layers	1
Number of units	10, 100, 190
Number of epochs	100
Number of features	93

Due to space constraints, only the *number of units* hyperparameter was considered to be explored, thus placing aside, in this work, the effects of other hyperparameters as

[3]https://github.com/tiagotschmidt/NAD_ML_models

learning rate, *number of layers* and *activation function*. The DL subject models were a simple MLP model, an LSTM model, and a CNN model. The DL models differ in unit types, repeated layer configurations, and final layer structures. All details of the implementation of the DL subject model are available in the public repository.

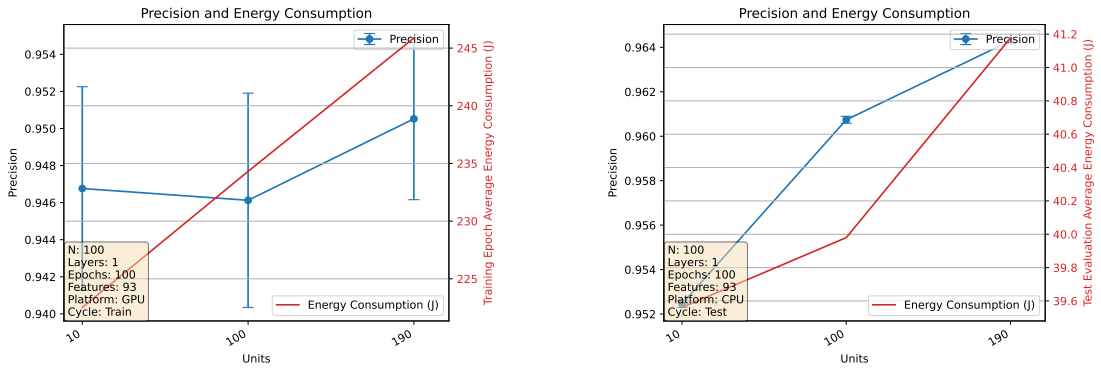
5.2. Experimental Results

The results are visualized in line plots that display the average values for *energy consumption* (y-axis) and *performance metric* (also y-axis) of different hyperparameter configurations (x-axis). The average value for the performance metric is accompanied by error bars that represent the standard error for that measurement. The error bar size is equivalent to 2 standard errors (with the average in the middle). Information about the remaining DL model configuration snapshot is displayed in a hover box.

The average energy consumption value displayed is the **average total energy consumption** of the **ML life cycle executed**. That is, for *training* profiles, the energy consumption value plotted is the **energy consumed** to **iterate over** 80% (due to the train/test split done in the `ExecutionEngine`) $\times$ **sampling rate** (S_r) $\times$ **total dataset entries**, repeating **number of epochs**. In other words, the value displayed in the graphs represents the **average total energy cost** to **execute the full training** routine. The same is applied to the inference results, but consider that the test routine does not repeat **number of epochs**.

5.2.1. MLP

Figure 3 displays the (*Precision*, *Energy Consumption*) X *Units* results for the MLP model. The other hyperparameters available were fixed, and their values are: 1 *layer*; 100 *epochs*; 100 *statistical samples*; 100% *sampling rate*.



(a) Training (b) Testing

Figure 3. MLP - (*Precision*, *Energy Consumption*) X *Units* Results.

The results display a behavior that was observed in all model results. Whenever the model complexity increases (more units) the energy consumption rises. Another common aspect in all profiles generated is the difference, in terms of magnitude, for the energy consumption values in training *versus* inference. This is due to the training routine iterating *epochs* times over the original dataset.

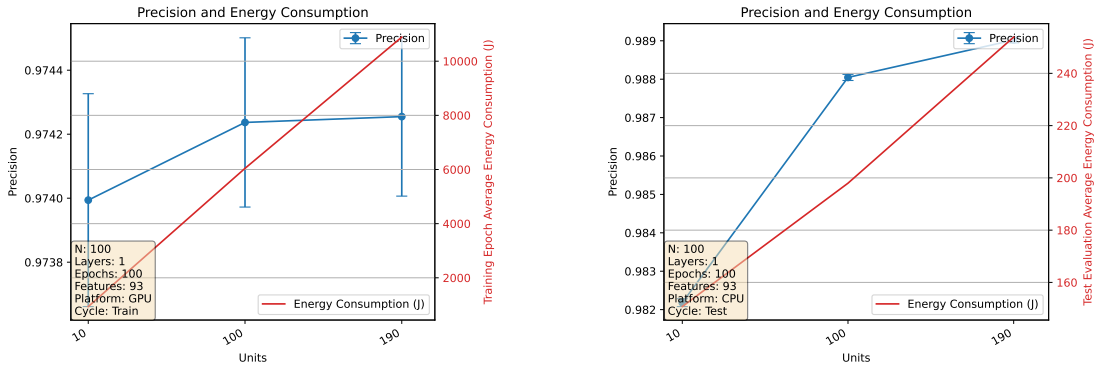
In terms of performance and energy trade-offs, the profiles outline a complex result. For the training profile, an increase in units is not linearly related to an increase in

performance. Instead, the response to *number of units* increase is uncertain and can even be considered a worse result in the medium-size configuration (100) units during training. More specifically, during training, the average precision value for the medium-size model (100 units) is worse than the small-size model (10 units). When the error is considered, clearly the performance difference between the small-size model and the medium-size model is uncertain. For the large configuration (190 units), the average precision increases, and the error bars are shifted upwards. Meanwhile, the energy measurement for the models during training is definitive: an increase of roughly 10 J per 90 units added.

For the inference profile, the precision contradicts training with consistent and low-variance results. The medium-size configuration is better in terms of precision, and the large-size configuration is still better than the small one. Additionally, the energy consumption cost of this rise in precision, from the small size configuration to the large size configuration, is only 1.6 J.

5.2.2. LSTM

Figure 4 displays the (*Precision*, *Energy Consumption*) X *Units* results for the LSTM model. The configuration snapshot repeats the MLP setup described above. Again, the linear relationship of energy consumption with units and differences in the order of magnitude for energy values can also be observed in Figure 4. In the LSTM profiles, it is possible to observe a great difference in the energy consumption values when compared to the MLP profiles, both in training and testing. This is due to a difference in complexity in the unit of the DL model: the *perceptron* when compared to the LSTM cell, is a far simpler unit.



(a) Training (b) Testing

Figure 4. LSTM - (*Precision*, *Energy Consumption*) X *Units* Results.

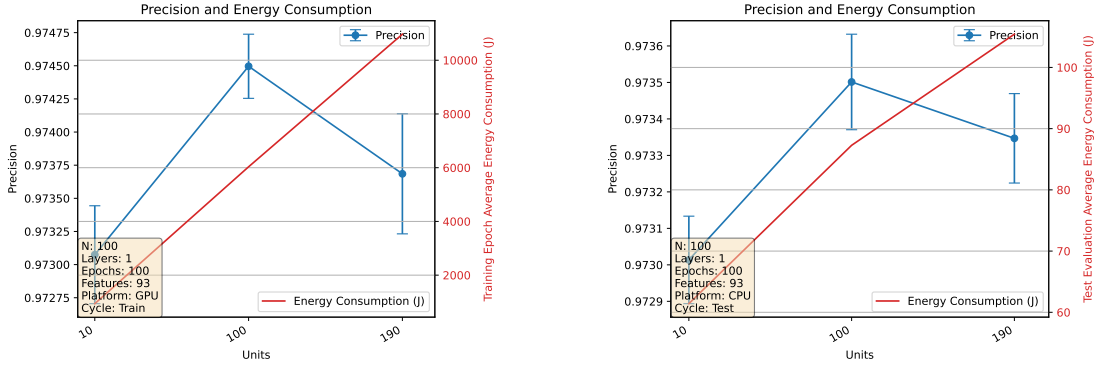
During training, the performance in general improves with more units. However, when the measurement errors are considered, the medium and large-size models have performed equally. The only arguable improvement is from the small-size model to the medium-size, as the average and error bars shift upwards.

During inference, the performance improvement is confirmed with low variance. At the same time, the energy cost difference between the small and large models is considerable: more than 80 J. This is even more expressive when compared to the energy cost difference observed in the two models for MLP (1.6 J). In general, the LSTM model performs better with a more complex model (more units). However, when the energy val-

ues are taken into account with precision, the trade-offs are costly: the increase of 0.7% precision also led to an increase of about 80 J during inference.

5.2.3. CNN

Finally, Figure 5 displays the (*Precision*, *Energy Consumption*) X *Units* results for the CNN model. The configuration snapshot repeats the MLP setup described earlier. Once more, the linear relationship of energy and *number of units* and the difference in order of magnitude for the testing and training energy values are present.



(a) Training (b) Testing

Figure 5. CNN - (*Precision*, *Energy Consumption*) X *Units* Results.

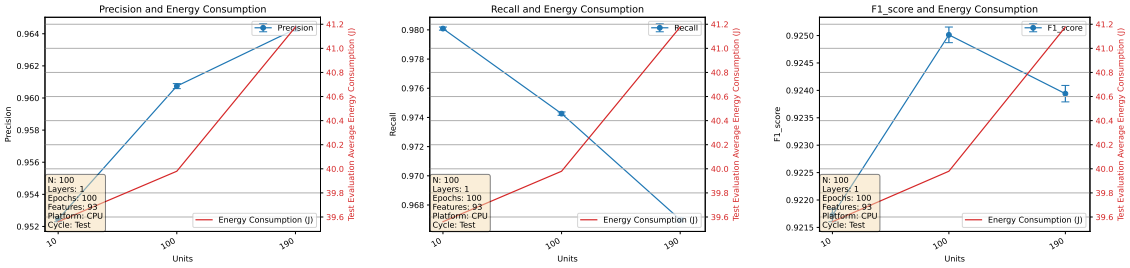
A different aspect of the CNN profiles is that the high variance of the training results repeats during inference. In particular, given that the CNN model is not particularly targeted to the network anomaly detection problem, this led to much more uncertain results (with higher error for the measurements). Another side effect was the energy cost: given that the structure and premises of a CNN model were not ideal for a network anomaly detection model, we observed an overall higher cost of energy (training and inference) and common performance levels.

Although the high standard error remains, the general profile for testing and inference is the same. The best configuration, in terms of averages only, is the medium-size configuration. Another advantage of the inference profile over the training one is the greater measurement consistency, which increased as seen by the difference in the error bars (with the training profile having larger error bars).

5.3. Discussion

The previous results provide important information to compare the three DL models in terms of performance and energy consumption. For example, an ML engineer that has a requirement of 95% precision would probably choose the MLP model with 10 *units*, as it consumes the least amount of energy (39.6 J) to perform inference (that is, to execute inference 125,973 times). In comparison, the minimum number of *units* for the LSTM model to reach 95% precision is also 10 *units*, but with an energy consumption of almost 160 J. Finally, the CNN model obtains a precision of 95% also with only 10 units, and an energy consumption of about 60 J.

Finally, the last discussion provided by the profiles generated on the experiment is the comparison of the *recall*, *precision*, and *f1-score* when the *number of units* is adjusted in one model. Due to space constraints, we will only consider the values in the inference cycle for the MLP model. Figure 6 displays the tendencies for precision, recall, and F1 score. Overall, the precision and F1 score values tend to positively respond to an increase in the *number of units*. On the other hand, for the large model size, the recall value shows signs of overfitting, leading to a minimal decrease in the recall value. However, even with this small decrease, the general F1 score of the model is higher with 190 units than with 10 units. Meanwhile, the energy cost to raise the F1-score from 0.9215 to 0.9240 is about 1 J only.



(a) Precision (b) Recall (c) F1-score

Figure 6. MLP - Precision Recall & F1-score Results.

6. Concluding Remarks

This paper proposes an analysis of the trade-offs between energy and the performance of network anomaly detection based on DL. To provide a precise energy measurement, statistically valid data, and realistic simulation of DL life cycle routines, this work proposed a profiling framework. A prototype was implemented and experiments enabled us to visualize the different configurations concerning energy and performance results. In particular, it was possible to observe details regarding the energy and performance trade-offs for the DL models (even trade-offs among different performance metrics).

In future work, we intend to explore the impact of additional techniques for model complexity optimization (*e.g.*, quantization, pruning) in terms of performance and energy consumption. Moreover, another intended extension is to compare and evaluate traditional network anomaly detection systems [Faustini et al. 2017], such as the statistical or even classical machine learning approaches.

Acknowledgments

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and grant #88887.954253/2024-00, CNPq grants #311276/2021-0 and #405940/2022-0, and FAPESP grants #2020/05152-7, #2023/00673-7 and #2023/00764-2.

References

Arafa, Y., ElWazir, A., ElKanishy, A., Aly, Y., Elsayed, A., Badawy, A.-H., Chennupati, G., Eidenbenz, S., and Santhi, N. (2020). Verified instruction-level energy consumption measurement for nvidia gpus. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*, CF '20, page 60–70, New York, NY, USA. ACM.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bianco, S., Buzzelli, M., Ciocca, G., and Schettini, R. (2020). Neural architecture search for image saliency fusion. *Information Fusion*, 57:89–101.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):1–99.

Candelieri, A., Perego, R., and Archetti, F. (2021). Green machine learning via augmented gaussian processes and multi-information source optimization. *Soft Computing*, 25(19):12591–12603.

Demme, J. and Sethumadhavan, S. (2011). Rapid identification of architectural bottlenecks via precise event counting. *SIGARCH Comput. Archit. News*, 39(3):353–364.

Dhanabal, L. and Shantharajah, S. (2015). A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International journal of advanced research in computer and communication engineering*, 4(6):446–452.

Ene, C. (2023). Council post: 10.5 trillion reasons why we need a united response to cyber risk.

Faustini, P. H. A., Silva, A. S., Granville, L. Z., and Schaeffer-Filho, A. E. (2017). Emprego de nfv e aprendizagem por reforço para detectar e mitigar anomalias em redes definidas por software. In *Anais do XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Porto Alegre, RS, Brasil. SBC.

Goel, B., McKee, S., and Själander, M. (2012). *Techniques to Measure, Model, and Manage Power*, volume 87, pages 7–54.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hsu, C., Chang, S., Juan, D., Pan, J., Chen, Y., Wei, W., and Chang, S. (2018). MONAS: multi-objective neural architecture search using reinforcement learning. *CoRR*, abs/1806.10332.

Kim, J., Kim, J., Thi Thu, H. L., and Kim, H. (2016). Long short term memory recurrent neural network classifier for intrusion detection. In *2016 International Conference on Platform Technology and Service (PlatCon)*, pages 1–5.

Kostas, K. (2018). Anomaly detection in networks using machine learning. *Research Proposal*, 23:343.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Marnerides, A., Schaeffer-Filho, A., and Mauthe, A. (2014). Traffic anomaly diagnosis in internet backbone networks: A survey. *Computer Networks*, 73:224–243.

Naseer, S., Saleem, Y., Khalid, S., Bashir, M. K., Han, J., Iqbal, M. M., and Han, K. (2018). Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, 6:48231–48246.

Preuveneers, D., Tsingenopoulos, I., and Joosen, W. (2020). Resource usage and performance trade-offs for machine learning models in smart environments. *Sensors*, 20(4).

Rodrigues, C. F., Riley, G., and Luján, M. (2018). Synergy: An energy measurement and prediction framework for convolutional neural networks on jetson tx1. In *Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA)*, pages 375–382.

Saeed, M. M., Saeed, R. A., Azim, M. A., Ali, E. S., Mokhtar, R. A., and Khalifa, O. (2022). Green machine learning approach for qos improvement in cellular communications. In *2022 IEEE 2nd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, pages 523–528.

Santos da Silva, A., Wickboldt, J. A., Granville, L. Z., and Schaeffer-Filho, A. (2016). Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 27–35.

Sedjelmaci, H., Senouci, S. M., and Al-Bahri, M. (2016). A lightweight anomaly detection technique for low-resource iot devices: A game-theoretic methodology. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6.

Teoh, T. T., Chiew, G., Franco, E. J., Ng, P. C., Benjamin, M., and Goh, Y. J. (2018). Anomaly detection in cyber security attacks on networks using mlp deep learning. In *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, pages 1–5.

Tripp, C. E., Perr-Sauer, J., Gafur, J., Nag, A., Purkayastha, A., Zisman, S., and Bensen, E. A. (2024). Measuring the energy consumption and efficiency of deep neural networks: An empirical analysis and design recommendations.

Vikram, A. and Mohana (2020). Anomaly detection in network traffic using unsupervised machine learning approach. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 476–479.

Wang, S., Balarezo, J. F., Kandeepan, S., Al-Hourani, A., Chavez, K. G., and Rubinstein, B. (2021). Machine learning in network anomaly detection: A survey. *IEEE Access*, 9:152379–152396.

Weaver, V. M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., and Moore, S. (2012). Measuring energy and power with papi. In *2012 41st International Conference on Parallel Processing Workshops*, pages 262–268.

Yang, X., Hua, S., Shi, Y., Wang, H., Zhang, J., and Letaief, K. B. (2020). Sparse optimization for green edge ai inference. *Journal of Communications and Information Networks*, 5(1):1–15.