Towards Neural Hardware Search: Power Estimation of CNNs for GPGPUs with Dynamic Frequency Scaling

Christopher A. Metz
Institut of Computer Science
University of Bremen
Bremen, Germany
cmetz@uni-bremen.de

Mehran Goli
Institut of Computer Science
University of Bremen/DFKI GmbH
Bremen, Germany
mehran@uni-bremen.de

Rolf Drechsler
Institut of Computer Science
University of Bremen/DFKI GmbH
Bremen, Germany
drechsler@uni-bremen.de

ABSTRACT

Machine Learning (ML) algorithms are essential for emerging technologies such as autonomous driving and application-specific *Internet of Things* (IoT) devices. *Convolutional Neural Network* (CNN) is one of the major techniques used in such systems. This leads to leveraging ML accelerators like GPGPUs to meet the design constraints. However, GPGPUs have high power consumption, and selecting the most appropriate accelerator requires *Design Space Exploration* (DSE), which is usually time-consuming and needs high manual effort. *Neural Hardware Search* (NHS) is an upcoming approach to automate the DSE for Neural Networks. Therefore, automatic approaches for power, performance, and memory estimations are needed.

In this paper, we present a novel approach, enabling designers to fast and accurately estimate the power consumption of CNNs inferencing on GPGPUs with *Dynamic Frequency Scaling* (DFS) in the early stages of the design process. The proposed approach uses static analysis for feature extraction and Random Forest Tree regression analysis for predictive model generation. Experimental results demonstrate that our approach can predict the CNNs power consumption with a *Mean Absolute Percentage Error* (MAPE) of 5.03% compared to the actual hardware.

CCS CONCEPTS

• **Hardware** → *Platform power issues*; • **Computing methodologies** → *Classification and regression trees.*

KEYWORDS

Machine Learning, Power Estimation, Neural Hardware Search, CNN, GPGPU

ACM Reference Format:
Christopher A. Metz, Mehran Goli, and Rolf Drechsler. 2022. 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), September 12–13, 2022, Snowbird, UT, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3551901.3556481

MLCAD '22, September 12–13, 2022, Snowbird, UT, USA

ACM ISBN 978-1-4503-9486-4/22/09…$15.00
https://doi.org/10.1145/3551901.3556481

1 INTRODUCTION

Machine Learning (ML) has become very popular in almost all industries, ranging from manufacturing to scientific-, health- and security-related applications, as well as autonomous driving [11, 12]. Among the existing ML algorithms, CNN is the most popular ML algorithm for image recognition in autonomous driving [15]. However, CNNs need high numbers of operations (e.g., floating point). The convolutional layers are responsible for over 90% of the computation in a CNN and require massive parallel processing with potentially trillions of computations per second [8]. This leads to the usage of ML accelerators (e.g., GPGPU) to speed up both neural network training and inferencing.

GPGPUs require high energy consumption. For example, to achieve high performance, the *Summit* supercomputer uses 27,648 NVIDIA Volta GPUs, leading to high energy consumption where a power supply of 13 million watts is required [9]. While data centers or cloud systems can provide this high number of GPGPUs to perform *High-Performance Computing* (HPC), due to power consumption limitations, for most emerging technologies applications, only a limited number of GPGPUs with certain configurations w.r.t the power consumption is available. On the other hand, due to the emerging usage of ML algorithms in IoT devices, embedded and edge computing systems, the need for HPC has been significantly increasing. This leads to an increase in power consumption of almost all devices that are using ML algorithms (e.g., CNNs). Hence, finding a trade-off between power consumption and HPC is of utmost importance for such applications.

Performing *Design Space Exploration* (DSE) is highly important to find such a trade-off, especially for applications with a limited power supply like IoT and edge devices where unlimited power supply is unavailable. A promising power management technique that is widely used during DSE is *Dynamic Voltage and Frequency Scaling* (DVFS) or *Dynamic Frequency Scaling* (DFS). The DVFS technique changes the voltage/frequency during the processing of applications, while DFS only changes the frequency. Both energy consumption as well as performance, can be optimized with these techniques. While the DVFS/DFS techniques for CPU-based applications are well-developed, in the case of the GPGPU, the study started only a few years ago [17]. Moreover, [1] pointed out that DVFS techniques for CPU do not suit for GPGPUs. Consequently, for GPGPUs, new DVFS techniques need to be developed.

In order to support system designers in finding suitable devices, GPGPU simulators can be used for DSE. Those simulators deliver details of GPGPU kernels to understand the execution behavior on GPGPUs [23]. Therefore, they use performance counters and specific hardware details to estimate the execution time and power

consumption. They reach an accuracy between 10% to 20% compared to real hardware [24]. However, these simulators require a long execution time and are significantly slower than native execution on real hardware [23, 24].

To tackle this issue, several predictive models have been developed [4, 17–19]. They can be classified either as empirical or statistical studies. The first one relies on code analysis, while the second one on the program performance counter. Empirical approaches can be called a bottom-up approach and usually require detailed information about the GPGPU micro-architecture. The statistical approaches ignore the GPGPU architecture and treat it as a black box. These approaches take details of the application behavior and analyse the relationship between performance counters, GPGPU power consumption and runtime [17].

Although the aforementioned methods can help designers to build fewer prototypes during DSE and avoid costly design loops, they need detailed application behavior analysis and specific profiler and profiling settings to collect the necessary performance counter. Moreover, most of them do not support power prediction in the case of DFS.

In this paper, we present a fast and accurate predictive model considering the DFS ability of GPGPUs and straightforward collectible predictors that do not need specific profiling settings. Compared to other empirical studies, we do not need a detailed break-up of the GPGPU micro-architecture and treat most parts of the GPGPU architecture as a black box. Thus, we combine the advantages of both empirical and statistical approaches. Our predictive model achieves a MAPE of 5.03%.

In summary, the main contributions of the paper are as follows:

(1) A fast and accurate power estimation model considering DFS to perform NHS for various configurations of GPGPUs,
(2) evaluation of different machine learning techniques to obtain the best predictive model (i.e., Random Forest Tree regression analysis),
(3) evaluating the applicability and accuracy of the proposed approach in estimating power consumption of 30 standard CNNs for the Nvidia V100S GPGPU (which is one of the most used GPGPUs in data centers).

The rest of this paper is organized as follows: Section 2 gives an overview of previous works. The motivation of the work is presented in Section 3. The proposed methodology is introduced in Section 4. Experimental results to show the efficacy of our approach are presented in Section 5. Finally, the paper is concluded in Section 6.

2 RELATED WORK

Power and performance prediction is inevitable to perform NHS. Hence, several approaches have been introduced to achieve this goal. Since this work focuses on power prediction, in this section, we consider the most relevant state-of-the-art power prediction methods to our work.

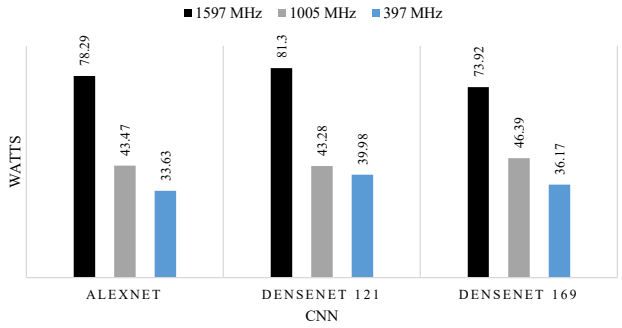
In general, power prediction methods can be divided into three main categories, which are 1) ML-based approaches that rely on learning the relationship between predictors (features) and output (i.e., power consumption) [7, 18, 20, 24], 2) simulation-based methods, which execute the application on virtual devices and measure the power consumption based on those simulations [2, 10] and 3) statistical approaches [4, 23], which use performance counters to model the relationship between an application and run-time profiling. Furthermore, most treat the micro-architecture of the GPGPU as a black box.

In [7], an ensemble ML-based approach is presented. Therefore, different ML-based models (e.g., ZeroR simple linear regression, *K-Nearest Neighbors* (KNN), bagging, random forest, sequential minimal optimization regression, decision tree, and neural networks) are created. Afterward, those ML-based models are unified in an ensemble model considering a weighting factor to emphasize those underlying models with the best prediction. However, this approach relies on performance counters, which are usually only available during the applications' execution and need specific profiling tools and settings. Subsequently, collecting training data and the input data for the final model is time-consuming and needs the exact profiler settings. This can limit the generality and applicability of the method. In contrast, our approach relies on easy-to-access predictors and less on dynamic runtime predictors. Moreover, the quality of the predicted results is specified as *Mean Absolute Error* (MAE) with the reported value 3.5% However, the error should be reported based on the MAPE as MAE is not a percentage value.

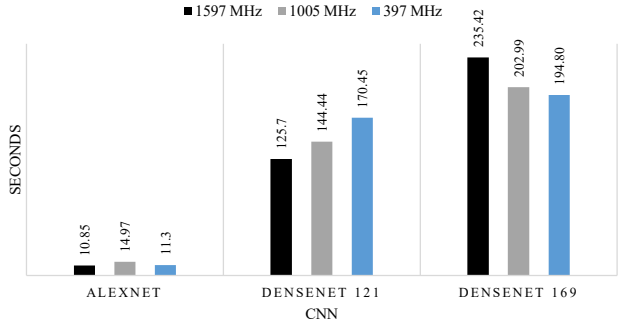
The approach introduced in [20] estimates the power consumption of CNNs for GPGPUs based on a KNN regression analysis. The prediction is based on hardware architectural details and PTX instructions. The PTX operations are counted and classified into different groups used as predictors. Furthermore, all hardware architectural details which affect the power consumption are considered. Due to manual feature selection, the most influential predictors are chosen. However, although the base frequency is a selected predictor, the approach does not consider DFS during the training data creation. Thus, it cannot be used in NHS with different configurations of a given GPGPU in terms of frequencies.

ALOHA [4] presents a statistical approach to evaluate CNNs for hardware-aware NAS on heterogeneous systems (e.g., GPGPUs and FPGAs). The operations, data transfers, and deployment of computing and communications resources can be estimated for a given device and CNN. This provides designers with essential information for hardware-aware NAS. However, we focus on NHS in this work. Thus, other details of the CNNs behavior on GPGPUs are essential. Moreover, this method requires a detailed and accurate execution model for different heterogeneous devices, which may not always be available.

Simulators like GPGPU-Sim [2] or GPU-ocelot [10] can execute CUDA applications on CPU systems. They simulate the behavior of GPGPUs and allow designers to measure the performance counter for a given GPGPU. Thus, they can be used to perform DSE as they provide designers with details of GPGPU kernels. They are versatile and, thus, can be applied to many applications. However, one major drawback of these simulators is the low execution speed. Since the CUDA code is executed on CPUs, the execution time is significantly slower than native execution on real GPGPUs. Moreover, considering a general solution to estimate the power consumption can reduce the final prediction result. This is mainly because applications from different domains have different features, characteristics, and behavior. Hence, a specified predictive model is preferable to a general predictive model.



(a) Power consumption on differen frequencies



(b) Run-time of various CNNs for differen frequencies inferencing on Nvidia V100S

Figure 1: Power consumption and Run-time of various CNNs inferencing for different frequencies on Nvidia V100S.

Overall, although the results of the aforementioned power prediction methods are complementary to our approach, they have some limitations in terms of speed, the need for detailed platform scheduling, as well as specific profiling tools and settings, and the lack of supporting NHS. Hence, the main goal of this work is to overcome these limitations.

3 MOTIVATION

A frequently used technique to reduce the power consumption of processing units like CPUs or GPGPUs is DVFS. However, the Nvidia Tools (e.g., nvidia-smi) only support frequency scaling and not voltage scaling [21]. Hence, we are referring to DFS in this work and not to DVFS.

Since the power consumption relies on the frequency (the higher the frequency, the more power is consumed), it can be reduced by scaling the frequency [17]. Consequently, the DFS technique needs to detect if the performance of a given application is affected by reducing the frequency of processing units.

Scaling the frequency of a GPGPU during CNN inferencing also affects the power consumption. In Fig. 1(a), three different CNNs are executed on the Nvidia V100S with frequencies of 1590MHz, 1005MHz, and 397Mhz, respectively. The results show that the power consumption is lower with smaller frequencies. However, the computational frequency has a different impact on the execution time of the three CNNs. As illustrated in Fig. 1(b), the run-time for Alexnet does not change significantly, while Densenet 169 is even

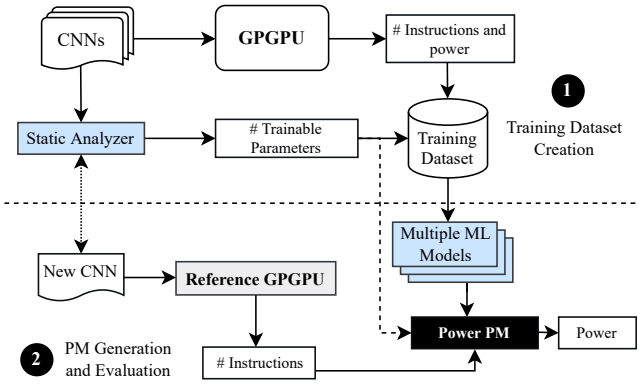


Figure 2: DFS-based power estimation methodology.

faster with lower frequencies. In the case of Densenet 121, lower frequencies lead to longer run-time.

Hence, system engineers must find the suitable GPGPU and frequency configuration for a given CNN. In a naive approach, designers need to execute CNNs for all available frequencies of a GPGPU and measure their power consumption to find the best trade-off. This way of exploring the design space for various CNNs is very time-consuming. For example, this process takes over 195 hours for 30 CNNs and all 196 available frequencies of the Nvidia V100S GPGPU. Hence, a fast and accurate approach can speed up the DSE process and significantly reduce the time-to-market constraints.

4 METHODOLOGY

The proposed methodology is structured into two main phases, which are 1) training dataset creation and 2) predictive model generation and evaluation. The overall flow of the proposed methodology is illustrated in Fig. 2. In the following, we explain each phase of the proposed approach in more detail.

4.1 Training Data Generation

An HPC Cluster with *Simple Linux Ressource Manager* (SLURM) is used for training data creation. The used machine is equipped with three Nvidia V100S 32GB, 256GB memory, and 2 AMD EPYC ROME 7272. Since the system is a computing cluster, the Home directory is a *Network Attached Storage* (NAS); a 10GBit/s ethernet connection connects it.

In order to take DFS into account, we used the functionality supplied by the *nvidia-smi* tool to set a fixed execution frequency [21]. This enables us to execute the CNN benchmarks on frequencies between 1597 MHz and 135 MHz on the Nvidia V100S GPGPU. Therefore, we program the SLURM-based HPC system (a batch script) to start a Job array for all combinations of available frequencies and CNNs. This program is illustrated in Fig. 3. To avoid any side effects, we ensure our benchmark is the only running job on the machine. This is done with the SLURM parameter *–exlusive* (Fig. 3, Line 5). The benchmark has three steps 1) setting the frequency, 2) executing the CNN and measuring the power consumption and execution time, and 3) resetting the frequency for the subsequent

```
1  #!/bin/bash
2  #
3  #SBATCH --job-name="Power_Benchmark"
4  ...
5  #SBATCH --exclusive
6  ...
7  declare -a CLOCK_RATES
8  CLOCK_RATES=(1597 ... 135)
9
10 declare -a CNNs
11 CNNs=('m-r50x1' ... 'alexnet')
12
13 declare -a combinations
14 index=1
15
16 for clock_rate in ${CLOCK_RATES[*]}
17 do
18  for CNN in ${CNNs[*]}
19  do
20   combinations[$index]="$clock_rate $CNN"
21   index=$((index+1))
22  done
23 done
24
25 parameters=(${combinations[${SLURM_ARRAY_TASK_ID}]})
26
27 clock_rate=${parameters[0]}
28 cnn=${parameters[1]}
29
30 nvidia-smi -pm 1 -i 0
31 nvidia-smi -i 0 -ac 1107,${clock_rate}
32 srun python3 benchmark.py -n ${cnn} -f ${clock_rate}
33 nvidia-smi -rac
34 nvidia-smi -pm 0 -i 0
```

Figure 3: Part of SLURM SBATCH Job script to execute the CNN benchmarks with different frequencies on the experimental setup.

execution (Lines 30 to 34). After the execution of CNN benchmarks on a given frequency, the measurements are added to a CSV file containing the GPGPUs name, CNN name, frequency, power consumption, and execution time. The observation is extended during further data collection steps for the predictors.

In order to get the exact number of executed instructions for CNN benchmarks, we use the Nvidia Profiler *nvprof*, which analyzes the CNN during execution [22]. Thus, we execute all 30 CNN benchmarks and create a profile for each of them. The extracted number of executed instructions extends the training dataset. The main reason is that the total number of PTX instructions cannot be extracted from abstract PTX files at compilation time as it does not contain dynamic information such as the length of loops or jump instructions based on the comparison (Fig. 4, Lines 14 and 15). Thus, at least one execution of the CNN benchmarks on real hardware or cycle-level simulators (which take much longer time than real devices) is required. In the *Training Dataset Creation* phase, the Nvidia V100S GPGPU is used to measure the number of executed instructions for each CNN.

In order to handle this issue for a new CNN in the second phase (Fig. 2, *PM Generation and Evaluation*), we take advantage of a reference GPGPU (which can be any available GPGPU) and the Nvidia profiler *nvprof* to obtain the total number of instructions.

Every CNN is different in terms of the number of neurons, the number of layers, activation function, and many more factors [13]. The number of trainable parameters is one option to measure the complexity of a neural network. A trainable parameter is a connection between two neurons that has weight. This weight changes during the training process. Considering the Bias-neurons, the trainable parameter between two fully connected layers TP_{ff} is defined as follows:

$$TP_{ff} = F_{-1} * F + F \tag{1}$$

Where F, F_{-1} are the number of neurons in the current layer and the number of neurons of the previous layer, respectively, in order to calculate the total number of trainable parameters TP_{total} for a Neural Network, all trainable parameters of each layer are summarized regardless of the type of neural network (e.g., Fully Connected, CNN, *Recurrent Neural Network* (RNN)).

$$TP_{total} = \sum_{i=1}^{n} TP_{i-1,i} \tag{2}$$

However, in the case of convolutional layers, the trainable parameter needs different calculations. There are two cases to cover:

- a convolutional layer connected to another convolutional layer TP_c,
- a convolutional or pooling layer connected to a fully connected layer TP_{cf}

For each convolutional layer connected to another convolutional layer, the calculation for TP_c is performed based on the following definition:

$$TP_c = K^2 * C * N + N \tag{3}$$

The parameters K, C, and N are details of the convolutional layers that stand for the kernel size (width), the number of channels, and the number of kernels, respectively. In the case that a convolutional or pooling layer is connected to a fully connected layer, the trainable parameters TP_{cf} are calculated as follows:

$$TP_{cf} = \begin{cases} O_{conv}^2 * N * F + F, \\ O_{pooling}^2 * N * F + F \end{cases} \tag{4}$$

$$O_{conv} = \frac{I - K + 2P}{S} + 1 \tag{5}$$

$$O_{pooling} = \frac{I - W}{S} + 1 \tag{6}$$

Here, the parameter O_{conv}, I, K, S and P are standing for the size (width) of the output image, the size (width) of the input image, the size (width) of the kernels used, the stride and the padding of a convolutional layer.

We calculate the trainable parameter (*Static Analyzer* module in Fig. 2, Phase 1) for all 30 benchmark CNNs and add the total number of trainable parameters for each CNN to the corresponding observation of the DFS benchmark. The final training dataset D is defined as follows:

$$D = \{d_i | d_i = \{y_i(c_i, tp_i, ins_i)\}; 0 < i < n\} \tag{7}$$

For each observation d_i the parameter c_i, tp_i, ins_i identify the GPGPUs' frequency, the CNN trainable parameter and the total number of executed PTX instructions, respectively. The output parameter y_i denotes the measured power consumption (in watts) for each CNN running on GPGPUs. The training dataset is split into 70%

```
1   // Generated by LLVM NVPTX Back-End
2   .version 6.0
3   .target sm_61
4   .address_size 64
5   ...
6   .reqntid 256, 1, 1{
7   .reg .pred %p<14>;
8   ...
9   mov.u32 %r13, %ctaid.x;
10  mov.u32 %r14, %tid.x;
11  shl.b32 %r15, %r13, 10;
12  shl.b32 %r16, %r14, 2;
13  or.b32 %r1, %r16, %r15;
14  setp.lt.u32 %p1, %r1, 718296;
15  @%p1 bra LBB0_2;
16  bra.uni LBB0_1;
17  LBB0_2:
18  ld.param.u64 %rd10, [fusion_135_param_0];
19  ...
20  LBB0_1:
21  ret;}
22  ...
```

Figure 4: Part of the PTX file of a CNN model.

training and 30% evaluation, which are independent and have no overlapping data. Afterward, the different predictive models (we use five different ML techniques) are trained on the dataset and evaluated in terms of accuracy and speed.

4.2 Predictiv Model Generation and Evaluation

We consider five common ML techniques for regression analysis, namely 1) KNN, 2) Decision Tree, 3) Random Forest Trees, 4) *eXtreme Gradient Boosting* (XGBoost) and 5) Linear Regression.

KNN is a non-parametric ML algorithm for clustering. Although it is designed for classification tasks, it can be used for regression analysis, too [14]. Decision Tree, Random Forest Tree, and eXtreme Gradient Boosting are tree-based decision algorithms that are also usually used for classification, but can also be used for regression analysis. The Decision Tree technique builds a binary tree structure, where a predictor defines each node and threshold [16]. The Random Forest Tree is an extension of this simple Decision Tree technique and consists of an collection of Decision Trees that creates a "Forest". Hence, the Random Forest Tree usually has better results than a single Decision Tree [3]. The XGBoost is a technique to speed up the runtime of tree-based ML techniques. Consequently, the execution time of XGBoost models is faster than for other tree-based ML algorithms [5]. However, to find the technique that provides the best predictive model, we train five different models and compare them in terms of accuracy and speed.

In order to compare different ML techniques and evaluate the generated predictive models, we use the MAPE and the R^2 coefficient. This ensures that the accuracy of all predictive models is calculated using the identical metric, providing a basis for comparison. We calculate the MAPE based on Eq. (8) and R^2 based on Eq. (9).

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{8}$$

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{9}$$

Here, y_i, $\hat{y}_i$, and $\overline{y}_i$ identify the observation value, the prediction value, and the average value of the output, respectively. Moreover, this also enables us to compare our results to state-of-the-art works.

5 EXPERIMENTAL RESULTS

The initial experimental results sound promising. We evaluate five commonly used ML-algorithm to build a predictive model for power consumption considering DFS. The experimental results are consolidated in Table 1, which shows the accuracy of the predictive models for average, top 10%, and bottom 10% prediction. The Linear Regression shows the worst result with a MAPE of 15.31% followed by the KNN with a MAPE of 7.74%. The tree-based ML algorithms, Decision Tree, XGBoost, and Random Forest Tree attained better results with MAPE of 6.03%, 5.43%, and 5.03%, respectively. The best predictive model is based on the Random Forest Tree algorithm, where the total number of instructions and the fixed frequency are used as predictors. In this case, the Random Forest Tree predictive model achieves a MAPE of 0.3% for the top 10% and 5.56% for the bottom 10% prediction. Since the features are simple to collect and consist of only two elements, the proposed approach can be easily applied and used. This can significantly enhance the DSE process and avoid the heavy tasks of specific setup and configuration, as well as the extraction of a large number of features that are usually required by most other approaches.

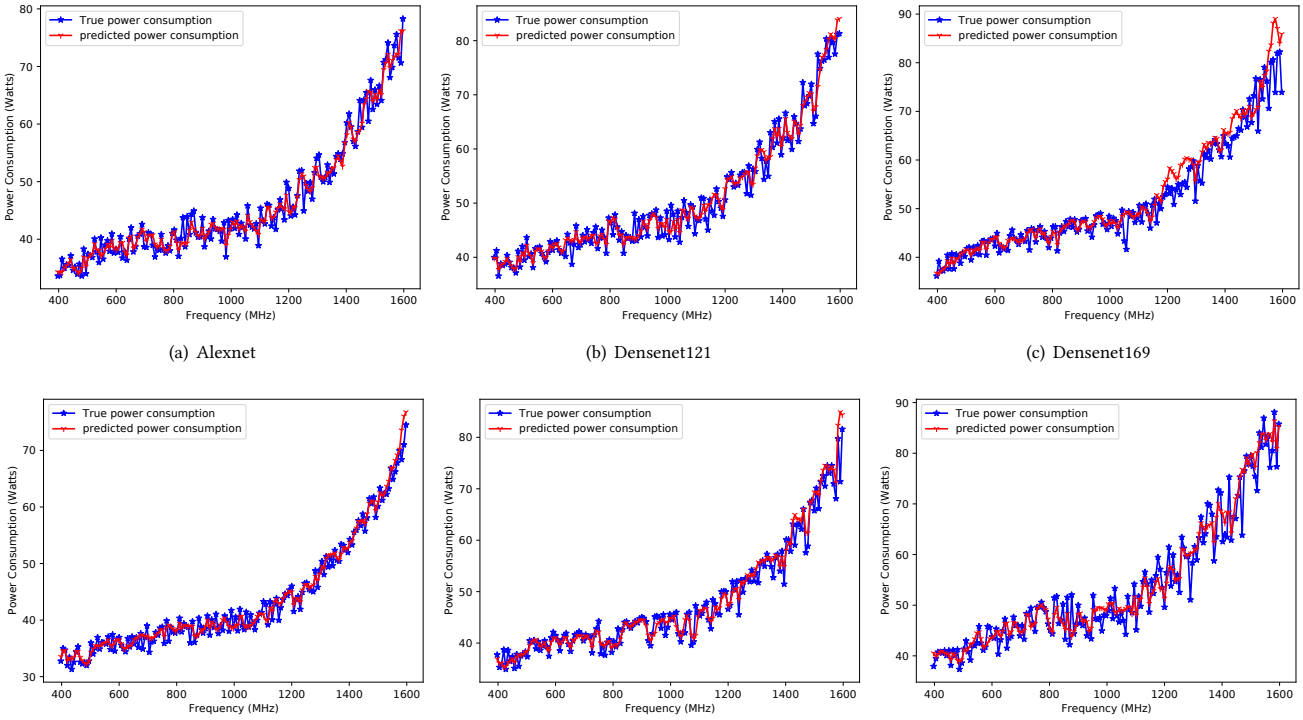
Compared to the state-of-the-art approach [20] with a MAPE of 8.3%, our proposed approach achieves *1.65*× better accuracy. The main reason is that the proposed approach is specified to a single GPGPU model and can predict the power consumption for various CNNs for this specific GPGPU. Please note that the GPGPU can be changed, and any other GPGPUs can be added to the training dataset. In this case, based on which GPGPU is considered as the target device, the predictive model can be adopted. Therefore, any extension on the training dataset can be quickly established.

The execution time of our approach is 0.0109 seconds in the worst case. Based on this, DSE for 30 different CNNs and 196 possible frequencies of the Nvidia V100S take about $30 * 196 * 0.0109s \approx 64.1s$ which leads to a tremendous speed-up of *11009*× in comparison to the naive approach. Moreover, it also opens another possible use case; online dynamic frequency scaling of GPGPUs. The predictive model can be used to estimate the power consumption of a CNN online on the device and, thus, to scale the frequency depending on the executed CNN in production. This opens additional power-saving options for systems executing different CNNs on the same device (e.g., GPGPU) like HPC-Systems or cloud providers. For example, the Nvidia V100S has 196 frequencies between 1597 MHz and 135 MHz that can be configured. Consequently, finding the frequency with the lowest power consumption requires 196 execution of the predictive model, which takes $196 * 0.0109s = 2.1364s$. In the case of the model's periodic executions, the best frequency can be calculated and cached after the first inferencing and used in an additional execution.

A comparison between the predicted power consumption, based on our predictive model and the actual power consumption on a real

Table 1: Comparison of four different ML-Regression algorithms in terms of accuracy and execution time

GPGPU	Regression Model	Accuracy					Execution Time	
		Average		Max Absolute Error	Top 10%	Bottom 10%	fast	slow
		MAPE	R^2		MAPE	MAPE		
V100S	Linear Regression	15.31%	0.6447	117.05 Watts	1.06%	16.88%	$3.5099e^{-5}$	0.0002
	K-Nearest Neighbors	07.74%	0.8027	93.09 Watts	0.52%	8.53%	0.0003	0.0017
	Random Forest Tree	**05.03%**	**0.9561**	**38.24 Watts**	**0.30%**	**5.56%**	**0.0050**	**0.0109**
	Decision Tree	06.03%	0.9359	38.38 Watts	0.32%	6.65%	$4.219e^{-5}$	0.0002
	XG Boost	05.43%	0.9512	48.13 Watts	0.34%	5.99%	$9.0800e^{-5}$	0.0003



(a) Alexnet (b) Densenet121 (c) Densenet169

(d) Efficientnetb0 (e) Efficientnetb1 (f) Efficientnetb2

Figure 5: Comparison of predicted and real power consumption for six different CNNs for frequencies between 397MHz and 1590MHz on the Nvidia V100S GPGPU.

device (e.g., Nvidia V100S) is illustrated in Fig. 5. This figure shows that the predictive model is close to the actual power consumption.

6 CONCLUSION

In this paper, we proposed a novel ML-based approach to estimate the power consumption of CNNs for GPGPUs with DFS. We illustrated how the power consumption of CNNs for a given GPGPU with various frequencies can be estimated by only considering two main features as predictors, namely the total number of executed instructions and trainable parameters (related to the CNNs topology). The proposed approach empowers designers to apply NHS and hardware-aware NAS, considering the power consumption of CNNs for GPGPUs. Our model can predict the power consumption of various GPGPU frequency configurations without retraining the model. Experimental results sound promising, and we believe this new line of research is helpful for making power estimation CNNs for GPGPUs with DFS truly a cross-cutting activity in the early stages of the design process.

As part of our future work, we plan to study automatic NHS and hardware-aware NAS based on our predictive models. Therefore, we will combine predictive models for power consumption estimation with a predictive model for performance estimation and build an NHS system. Furthermore, we will investigate other relevant predictors for CNNs such as FLOPs, *Multiply-Accumulated* (MAC) operations, or consider Neural Networks in ONNX [6] format. In order to harden our predictive model, we work on more standard CNNs and variations of these as well as GPGPUs.

ACKNOWLEDGMENTS

This work was partly supported by the German Federal Ministry of Education and Research (BMBF) within the project VerSys under contract no. 01IW19001, the Data Science Center of the University of Bremen (DSC@UB), and the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

REFERENCES

[1] Yuki Abe, Hiroshi Sasaki, Martin Peres, Koji Inoue, Kazuaki Murakami, and Shinpei Kato. 2012. Power and Performance Analysis of GPU-Accelerated Systems. In *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*. USENIX Association, Hollywood, CA. https://www.usenix.org/conference/hotpower12/workshop-program/presentation/Abe

[2] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *2009 IEEE international symposium on performance analysis of systems and software*. 163–174.

[3] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1, 5–32.

[4] Paola Busia, Svetlana Minakova, Todor Stefanov, Luigi Raffo, and Paolo Meloni. 2021. ALOHA: A Unified Platform-Aware Evaluation Method for CNNs Execution on Heterogeneous Systems at the Edge. *IEEE Access* 9, 133289–133308.

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[6] ONNX Community. 2019. ONNX. https://onnx.ai/.

[7] Bishwajit Dutta, Vignesh Adhinarayanan, and Wu-chun Feng. 2018. GPU Power Prediction via Ensemble Machine Learning for DVFS Space Exploration. In *ACM International Conference on Computing Frontiers*. 240–243.

[8] M. Fingeroff. 2021. Machine learning at the edge: using HLS to optimize power and performance. https://go.mentor.com/5_3ik. In *The Mentor - A Siemens Business (white paper)*.

[9] Fernanda Foertter. 2018. Summit GPU Supercomputer Enables Smarter Science. https://developer.nvidia.com/blog/summit-gpu-supercomputer-enables-smarter-science/. Accessed: 22.03.2022.

[10] gatech. 2013. GPU Ocelot. https://gpuocelot.gatech.edu/doxygen/.

[11] Mehran Goli and Rolf Drechsler. 2020. PREASC: Automatic portion resilience evaluation for approximating SystemC-Based designs using regression analysis techniques. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 25, 5 (2020), 1–28.

[12] Mehran Goli, Jannis Stoppe, and Rolf Drechsler. 2018. Resilience evaluation for approximating SystemC designs using machine learning techniques. In *2018 International Symposium on Rapid System Prototyping (RSP)*. IEEE, 97–103.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[14] Sadegh Bafandeh Imandoust, Mohammad Bolandraftar, et al. 2013. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International journal of engineering research and application* 3, 5, 605–610.

[15] Seung-Wook Kim, Keunsoo Ko, Haneul Ko, and Victor CM Leung. 2021. Edge-Network-Assisted Real-Time Object Detection Framework for Autonomous Driving. *IEEE Network* 35, 1, 177–183.

[16] Ronny Kohavi and J. Ross Quinlan. 2002. *Data Mining Tasks and Methods: Classification: Decision-Tree Discovery*. Oxford University Press, Inc., 267–276.

[17] Xinxin Mei, Qiang Wang, and Xiaowen Chu. 2017. A survey and measurement study of GPU DVFS on energy conservation. *Digital Communications and Networks*, 89–100.

[18] Christopher A. Metz, Mehran Goli, and Rolf Drechsler. 2021. Early Power Estimation of CUDA-Based CNNs on GPGPUs: Work-in-Progress. In *International Conference on Hardware/Software Codesign and System Synthesis*. 29–30.

[19] Christopher A. Metz, Mehran Goli, and Rolf Drechsler. 2021. Pick the Right Edge Device: Towards Power and Performance Estimation of CUDA-based CNNs on GPGPUs. *CoRR* abs/2102.02645. arXiv:2102.02645

[20] Christopher A. Metz, Mehran Goli, and Rolf Drechsler. 2022. ML-based Power Estimation of Convolutional Neural Networks on GPGUs. In *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. 166–171.

[21] Nvidia. 2016. nvidia-smi. https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf.

[22] Nvidia. 2022. Profiler User's Guide. https://docs.nvidia.com/cuda/profiler-users-guide/index.html. Accessed: 2022-02-02.

[23] Qiang Wang and Xiaowen Chu. 2020. GPGPU performance estimation with core and memory frequency scaling. *IEEE Transactions on Parallel and Distributed Systems*, 2865–2881.

[24] Gene Wu, Joseph L Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. In *IEEE International Symposium on High Performance Computer Architecture*. 564–576.