Performances Modeling of Computer Vision-based CNN on Edge GPUs

HALIMA BOUZIDI, Université Polytechnique Hauts-de-France, LAMIH/CNRS, France
HAMZA OUARNOUGHI, Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA Hauts-de-France, France
SMAIL NIAR, Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA Hauts-de-France, France
ABDESSAMAD AIT EL CADI, Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA Hauts-de-France, France

Convolutional Neural Networks (CNNs) are widely used in various fields nowadays, particularly for computer vision applications. Edge platforms have drawn tremendous attention from academia and industry due to their ability to improve execution time and preserve privacy. However, edge platforms struggle to satisfy CNN's needs due to their computation and energy constraints. It is then challenging to find the most efficient CNN that respects accuracy, time, energy, and memory footprint constraints for a target edge platform. Furthermore, given the size of the design space of CNNs and hardware platforms, performance evaluation of CNNs entails several challenges and efforts. Consequently, the designers need tools to quickly explore the large design space and select the CNN that offers the best performance trade-off for a set of hardware (HW) platforms. This paper proposes a Machine Learning (ML) based modeling approach for CNN performances on edge GPU-based platforms for vision applications. We implement and compare five (5) of the most successful ML algorithms for accurate and rapid CNN performances predictions on three (3) different edge GPUs in image classification. Experimental results demonstrate the robustness and usefulness of our proposed methodology. For three of the five ML algorithms, namely XGBoost, Random Forest, and Ridge Polynomial regression, an average error of 11%, 6%, and 8%, have been obtained for CNN inference execution time, power consumption, and memory usage, respectively.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; **Machine learning algorithms**; • **Computer systems organization** → **Embedded hardware**; • **Hardware** → *Power estimation and optimization.*

Additional Key Words and Phrases: Performance Modeling, CNN, Edge GPU, Execution Time, Power Consumption, Memory Usage, Machine Learning, Regression Analysis

1 INTRODUCTION

Machine Learning (ML) algorithms demonstrated their efficiency in several application domains such as Computer Vision (CV) and Natural Language Processing (NLP). Powerful hardware (HW) platforms combined with large and varied datasets allow ML algorithms to address complex issues in various use cases, such as autonomous driving or healthcare. As a part of ML, many Deep Learning (DL) algorithms, such as Convolutional Neural Networks (CNN), outperform state-of-the-art approaches in CV applications. For instance, in autonomous driving where

Authors' addresses: Halima Bouzidi, Université Polytechnique Hauts-de-France, LAMIH/CNRS, Valenciennes, France, Halima.Bouzidi@uphf.fr; Hamza Ouarnoughi, Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA Hauts-de-France, Valenciennes, France, Hamza.Ouarnoughi@uphf.fr; Smail Niar, Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA Hauts-de-France, Valenciennes, France, Smail.Niar@uphf.fr; Abdessamad Ait El Cadi, Université Polytechnique Hauts-de-France, LAMIH/CNRS, INSA Hauts-de-France, Valenciennes, France, Abdessamad.AitElCadi@uphf.fr.

1539-9087/2022/3-ART $15.00
https://doi.org/10.1145/3527169

more than 85% of applications are based on CV, CNNs are widely used for scene understanding and perception [29].

Processing data near sensors makes edge computing mandatory for privacy and network congestion concerns. CNNs can also take advantage of the opportunities provided by edge computing. Furthermore, several HW platforms have been proposed to support CNNs in the edge. For this purpose, Application Specific Integrated Circuits (ASIC), Field Programmable Gate-Arrays (FPGA), Graphics Processing Units (GPU), or Micro-Controller Units (MCU) have been widely used in edge computing for CNNs.

GPUs exploit massive data parallelism but consume a significant amount of power. However, GPU's companies, such as NVIDIA, have recently designed promising technologies for edge applications. For instance, Nvidia edge GPUs used in this paper support multiple power modes. Nvidia Jetson Nano delivers up to 472 GFLOPS at 5W. Compared to other HW platforms, GPUs also offer fast and flexible product development and short time-to-market. For this reason, they represent attractive solutions for companies interested in flexible DL-based systems [30, 35].

As CNNs are becoming more accurate, and then more complex, edge platforms struggle to deal with CNNs heavy computation and memory workload. With all advances in edge computing, finding the best trade-off between CNNs accuracy and their HW performances regarding latency, energy efficiency and resource utilization is still challenging. Finding the best combination between CNNs models and HW platforms is a tedious task [12] that needs a long time to explore the large and complex design space. State-of-the-art approaches propose to model CNNs as a computation workload represented by their FLoating-point OPerations (FLOPs) and memory usage. It has been demonstrated that CNNs' performances are not exclusively correlated with their FLOPs or memory consumption [9]. In this paper, the term *performances* refers to the three (3) CNN metrics: Inference execution time, power consumption and memory usage. Consequently, in our work, the term maximizing performances means reducing execution time, power consumption and memory usage.

Rapid CNN performance estimation becomes crucial to reduce design time and efficiently implement CNN on edge platforms. Early CNN performance estimation helps quickly determine the best CNN implementation and the most efficient corresponding HW. This approach can either help to choose the best CNN for a given HW platform or in exploring different HW platforms for a specific CNN. It can also be used as a surrogate-based model to guide the optimization process to the most promising CNN/HW architectures.

This paper proposes a methodology that helps HW and CNN designers choose the most efficient CNN while considering different edge GPUs based on early and rapid performances estimation. We believe that a good performance predictor must have the following features:

- It must be accurate and provide performances estimations rapidly.
- It must be highly flexible to be easily adapted to different CNNs and edge GPUs.
- If any, the number of the model hyperparameters must be reduced and easy to tune.

In this work, we implement and compare five (5) of the most frequently used state-of-the-art prediction algorithms to estimate the CNN's inference performances on three (3) different NVIDIA edge GPUs. As these algorithms have a set of parameters and hyper-parameters, they are denoted *Models* in the rest of the paper.

The following ML-based models have been used as performance predictors: Multiple Linear Regression using Ridge Polynomial Regression (denoted *Poly*), Multi-Layer Perceptrons (denoted *MLP*), Support Vector Regression (denoted *SVR*), Random Forest (denoted *RF*) and eXtreme Gradient Boosting (denoted *XGBoost*). The aim of our work is to compare the five (5) abovementioned models by analyzing their strengths and weaknesses regarding CNN performances prediction. For these models, we analyze:

(1) The performance of the prediction models by measuring the Mean Absolute Percentage Error (MAPE). We also assess the generalization power of the models regarding new and unseen CNN architectures (see subsections 5.2.1, 5.3.1 and 5.4.1).

(2) The time to tune and train the ML-based prediction models, as well as the the time to run the prediction models on different HW platforms (see subsections 5.2.2, 5.3.2 and 5.4.2).
(3) Finally, the capacity of the prediction models to preserve the rank of the estimations regarding the rank of the measured values on the real HW platforms (see subsections 5.2.3, 5.3.3 and 5.4.3).

The resulting performance prediction models have been evaluated on different test data. Moreover, we validate the portability of the proposed models on three NVIDIA edge GPUs: Jetson Nano, TX2 and AGX Xavier. Finally, our modeling methodology can be extended to other edge GPUs since it relies on a high-level characterization of CNNs independently from the HW platform.

Our experimental results show an average prediction error of 11%, 6% and 8%, for CNN inference execution time, power consumption and memory usage, respectively, on the three edge GPUs. We have also noticed a strong rank correlation between the measured and predicted performance metrics. Furthermore, the studied models demonstrated their efficiency regarding prediction error, rank-preserving, tuning, training costs and prediction latency. This makes the models suitable for real-time scenarios when the trade-off between the abovementioned factors is highly desired.

Our paper is structured as follows. Section 2 gives a literature review of CNN performance estimation and design space exploration. In section 3, we first analyze and discuss the CNN features that should be considered for performances prediction, then a survey of the used ML-based approaches is presented. The followed evaluation methodology and the used experimental setup are detailed in Section 4. Section 5.1 presents experimental results. Then we discuss results, limitations and future work of our approach in 6. Finally, conclusion will be given in section 7.

2 RELATED WORK

CNNs implementation on edge platforms has drawn several challenges. Considering both accuracy and execution efficiency is mandatory for optimal deployment of CNN on edge platforms. Therefore, the designer must maximize the CNN accuracy while minimizing its execution time, power consumption and memory usage. In this context, many solutions have been proposed in order to consider these metrics when designing CNNs for the edge. This section presents the most relevant related works going from handcrafted CNNs, to performance profiling and modeling.

2.1 Handcrafted CNN

These strategies are based on designing CNNs that are particularly tailored for the edge [19, 60, 67, 68]. However, these techniques aim to manually reduce the overall computational load and the memory footprint. Thus, the performances of the resulting CNNs may vary depending on the hardware platform as they are not designed for specific hardware. Other model-level optimization techniques have been widely investigated such as quantization [18, 51] and pruning [38, 39, 73].

2.2 Benchmarking and performances analysis

This involves implementing and executing CNNs on a target edge platform then measuring and analyzing their performances to investigate further optimization techniques [28, 36, 52, 63, 66, 69]. However, this solution entails significant efforts given the complexity of the deployment process and the diversity of CNNs and edge platforms. To overcome this issue, it is needed to characterize the CNN performances to infer a generalized model capable of predicting the performances of any CNN. In the literature and in our paper, this kind of approach is called *performances modeling*.

2.3 Performances modeling

In the following, we review existing performance modeling approaches, classified according to the target performance metric.

2.3.1 Execution time modeling. CNN inference execution time is an important metric for real-time applications with hard constraints, such as autonomous driving. Thus, it is necessary to consider the execution time when designing CNN for this kind of systems.

In the literature, models are used either to predict CNN training and/or inference execution time. In [4], the authors characterize inference execution time on GPU platforms by using different ML-based approaches: linear regression, SVR and RF with a Bulk Synchronous Parallel (BSP) based analytical model. They exploit profiling results obtained from nine (9) benchmarks executed on nine (9) different GPUs. However, the proposed approach considers general purpose applications that cannot be adapted to DL applications.

In [65], the authors propose analytical models to characterize DL training workloads in large computing clusters. Using different training hyperparameters, the authors identify performance bottlenecks for various DNN workloads. For instance, updating CNN weights and gradients consumes almost 62% of the total execution time on average for the training phase.

Authors in [55] propose a set of prediction models for training time, with Stochastic Gradient Descent (SGD) optimization. They target distributed GPUs in four (4) different DL frameworks. The authors analytically modeled the overall training time as the sum of the communication time between GPU nodes, I/O processing time and GPU processing time. The resulting prediction models have been used to compare and identify the performance bottleneck of the four (4) frameworks to further optimize them according to the workload features. However, the authors didn't provide a detailed description of the considered features to model the GPU processing time. Furthermore, they only evaluated their prediction models on three CNNs: AlexNet, GoogleNet and ResNet50.

PALEO [50] is another tool for estimating training execution time. In PALEO, the number of Floating-point OPerations (FLOPs) required for an epoch is multiplied by a scaling factor to obtain the training phase's execution time. However, PALEO does not consider numerous other operations that do not scale linearly with the number of FLOPs and significantly impact execution time. These works are complementary to ours as here we focus on inference performance.

Authors in [10, 31, 57, 64] leverage ML approaches to model CNN inference execution time on a layer-wise granularity. Contrary to these works, we propose CNN performances modeling on model-level granularity. Hence, providing more generalization for different types of CNN architectures. Layer-wise modeling approaches are not adapted to complex CNN architectures with dependencies between layers. For instance, ResNet [20, 21] and DenseNet [27] are characterized by skip and dense connections between layers. Another example is the parallelism in CNNs such as GoogleNet [59] where layers within the inception block can be evaluated simultaneously on the GPU [50]. These dependencies can therefore lead to higher prediction errors when estimating the overall performance of the CNN model [8].

In [44, 70], the authors analytically model the execution time of CNNs to find the best design on FPGA-based platforms. In [2], the authors propose a latency prediction model on FPGA, composed of a lookup table and a scheduler. The lookup table stores DNN operations and their corresponding latencies, where the scheduler maps operations to parallel units and calculates the overall latency. In our case, such an approach will lead to poor performance given the difficulty of implementing the scheduler part. Also, the Cuda scheduler of GPU is not open-source. Hence, it is hard to make precise assumptions on how DNN operations are scheduled on the GPU. Therefore, approaches based on analytical modeling are hard to implement for edge GPUs due to the complex interactions between the processing units (CPU and GPU).

2.3.2 Power and energy consumption modeling. Power consumption is an essential metric in edge computing. Indeed, minimizing power consumption is a critical factor for battery-based systems. For this reason, having a tool to evaluate the power and energy consumption of different CNNs and edge hardware architectures is important. In the following papers, as in our current work, both the static and dynamic energy or power consumption are considered in the estimation.

In [53], the authors propose a multi-variable linear regression to predict the energy consumption of the inference phase for CNNs based on the number of SIMD instructions and main memory accesses. The authors used Nvidia Jetson TX1 GPU and obtained about 20% of an average relative error. However, modeling CNN power consumption using only the features aforementioned is insufficient to obtain accurate estimation (see section 5.3). Other features such as CNN architectural dependencies must be taken into account.

The idea behind tools proposed in [10, 57, 64] is to explore the hyperparameter space of the most common layers of DNNs, such as convolution, pooling and fully-connected, to estimate their power consumption. Then, these papers estimate each layer's power consumption separately to provide an average power consumption for the whole DNN. However, it is difficult to capture the exact power consumption of each layer as its execution time is very short, leading to higher errors in the overall model power consumption prediction.

In [24] authors use regression-based performance estimators for both energy consumption and accuracy to explore the best quantization and scaling factors for DNNs on FPGA-based platforms. Unlike our work that uses ML to model the correlation between CNN workloads and power consumption, authors in [42, 43, 58] use analytical models to estimate the average power consumption of DNNs on FPGA and ASIC platforms. Nevertheless, analytical models lack flexibility as they require calculating low-level details of the CNN execution, such as memory transactions and processing elements utilization.

2.3.3 Memory usage modeling. Memory usage sets up a constraint on whether the CNN can be deployed on the target edge platform or not. Thus, it is necessary to minimize the memory needed to load the CNN and run the inference properly. In fact, the total amount of memory needed to run a single CNN inference comes from three primary sources: 1) parameters (i.e., weights and biases), 2) intermediate activations (i.e., features maps) and 3) the computation added by the DL framework. However, as detailed in section 3.2, calculating the total memory usage by measuring only the memory needed for CNN weights and activations is not accurate. The DL framework requires the most considerable amount of memory to run the inference. In our work, we estimate the total memory needed to run the CNN inference on the edge GPU by considering the memory allocated to the DL framework. [57] proposes to model memory usage from structural hyperparameters of the CNN, such as the number of hidden units. The authors use linear regression to approximate the memory prediction function for modeling. However, they have trained their prediction models on only variants of AlexNet [34]. In [41], the authors propose a modeling methodology for CNN memory usage on CPU and GPU platforms. Their approach is based on characterizing the memory requirements of convolutional and fully-connected layers to predict the entire CNN's memory usage.

Authors in [5, 33] propose prediction models of cache memory hierarchies for DNNs on modern discrete GPU platforms to investigate further optimization of memory performance on GPUs. There are also works that leverage performances estimators to reduce CNN memory footprint during training [11], or inference [40]. However, they didn't consider other performance metrics such as execution time and power consumption. Moreover, these works didn't target edge GPUs.

3 PROPOSED APPROACH

The ever-increasing complexity of CNN on edge platforms has made the co-optimization of accuracy and hardware efficiency extremely important. In this paper, we focus on studying computer vision-based CNNs on edge GPUs.

Figure 1 illustrates the complex correlation between the aforementioned metrics for state-of-the-art CNNs for image classification.

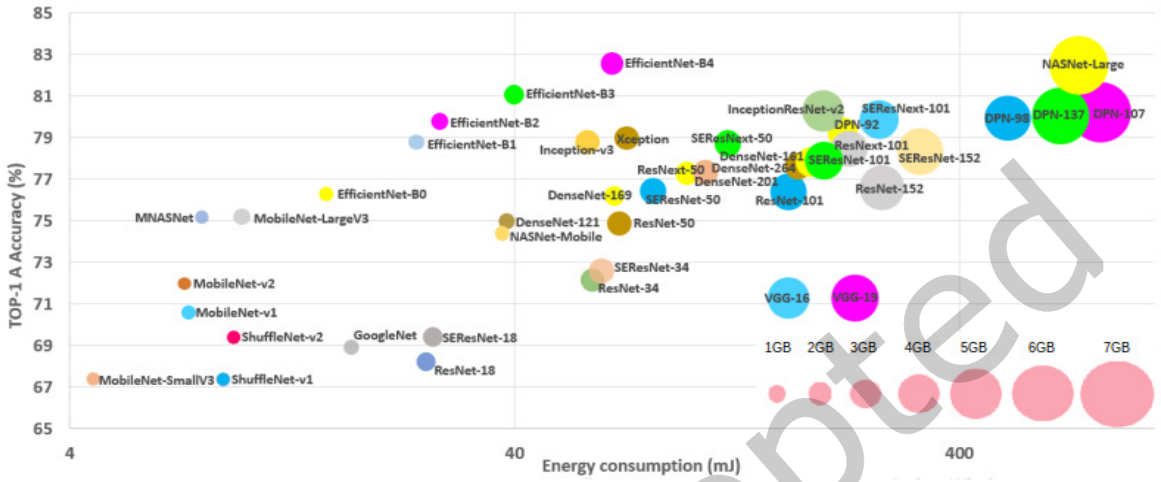

Fig. 1. Correlation between CNN accuracy, on ImageNet dataset, and their energy consumption and memory usage. The size of the circle corresponds to the total memory usage from 1 Gigabyte to 7 Gigabytes

In figure 1, each circle represents a CNN model. The size of the circles represents the maximum amount of memory required to run the CNN during the inference. Typically, the CNN memory usage reflects the memory used to hold the CNN parameters as well as the inference computations using the DL framework [41]. The x-axis shows the measured energy consumption calculated from the execution time and the power usage during the inference. The y-axis shows the TOP-1 accuracy of each CNN, obtained using the ImageNet dataset [17]. We note two main observations:

(1) The accuracy is not always correlated with either energy consumption or memory usage. For instance, EfficientNet-B4 [61] and NASNet-Large [72] both achieve about 83% TOP-1 accuracy. However, EfficientNet-B4 consumes ~89% less energy and requires ~53% less memory than NASNet-Large.
(2) The memory usage is often correlated with energy consumption. Thus, NASNet-Large , DPN-98, DPN-107 and DPN-137 [14], for instance, have significant memory footprints and consume a large amount of energy.

These observations confirm the complexity of finding the best trade off between CNNs requirements and edge platforms constraints. Moreover, an exhaustive exploration of CNNs on target hardware is tedious as the CNN performances evaluation is time-consuming. The exploration complexity becomes even worse in the case of multiple CNNs and multiple target edge platforms. Thus, one of the best ways to overcome this issue is to design accurate and reusable performance prediction models to estimate CNN performances on a target edge platform without executing the inference.

We propose in this paper a modeling methodology to design prediction models for CNN performances on edge GPUs. Ultimately, the goal is to use the elaborated prediction models within a multi-objective optimization process to explore the design space of CNN architectures and edge platforms. Hence, the prediction models can be easily re-used as objective functions to be optimized during the design exploration.

Our modeling approach is based on finding the correlation between the most important CNN features (e.g., image input size, number of hidden layers, neurons, activations, weights) and the performance metrics (i.e., execution time, power consumption and memory usage) on edge GPUs.

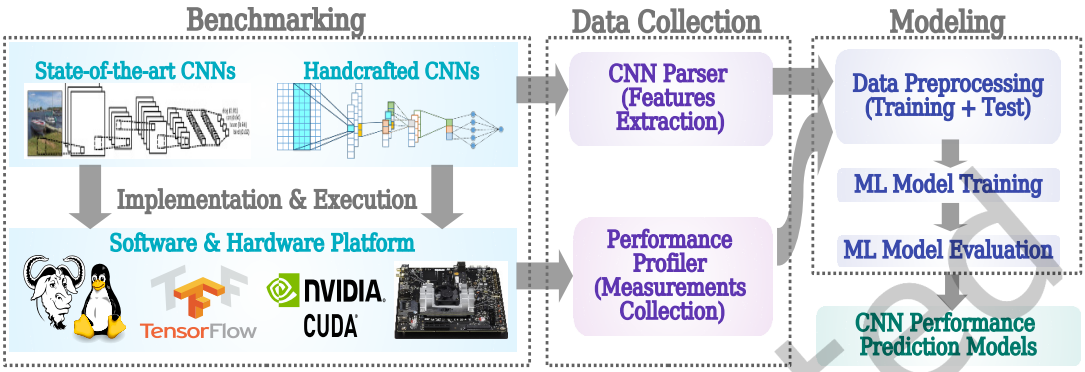

Fig. 2. Modeling methodology for CNNs performances prediction models

Our proposed methodology is depicted in figure 2 and is composed of three main steps.

(1) **Benchmarking** where we measure the execution performances of several CNNs on various edge GPUs. For this purpose, we establish the main CNN model-level features that impact the execution performance metrics. This impact is evaluated using different data analysis methods such as correlation and regression analysis.
(2) **Data Collection** where we collect two types of data from our benchmarks: Performance measurements and CNN features to be utilized as input prediction variables. These data are obtained using CNN architecture parser and performance profilers. On the one hand, the model parser takes a CNN architecture description as an input and outputs its detailed architectural features. On the other hand, the performances profiler takes a CNN architecture description, generates a ready-to-deploy CNN, deploys and executes the CNN on the target edge GPU, then returns the execution performance metrics values (execution time, power consumption and memory usage).
(3) **Modeling** which consists on elaborating a set of ML-based prediction models for each performance metric and edge GPU. In our paper, we design models for execution time, power consumption and memory usage. This step also involves the data pre-processing to perform feature transformation (i.e., data encoding and transformation) and feature selection for the ML-based prediction models. The prediction models are trained on the collected data from the previous step. The training process includes tuning the prediction model's hyperparameters and learning the model's internal parameters. Finally, we validate the generated prediction models for each performance metric and edge GPU by following the evaluation methodology detailed in section 4.

The next section presents the modeling problem formulation. Before detailing the modeling formulation, the table 1 summarizes the notation used in the rest of the paper.

3.1 Problem formulation

Predicting CNNs performance metrics on edge GPUs can be formulated as follows: As inputs, we have a CNN (cnn_i) characterized by a vector of n features $(f_1, f_2, \ldots, f_n)$. For instance, the number of convolutional and fully-connected layers, the input image size and the number of neurons are all considered CNN features. A

Table 1. Summary of notations

Name	Description
n	# of CNN input features
f_i	CNN input feature (e.g., FLOPs, number of layers, parameters..etc)
cnn_i	Vector of CNN input features
$edgpu_j$	Edge GPU indexed by j
k	Performance metric: execution time, power consumption, or memory usage
$\hat{y_{ijk}}$	Predicted value of the performance metric k of cnn_i on $edgpu_j$
y_{ijk}	Measured value of the performance metric k of cnn_i on $edgpu_j$
D_{jk}	Dataset of input features and measured values of k on $edgpu_j$

performance metric prediction function $\boldsymbol{T_{k,edgpu_j}}$ where $k \in \{$*execution time, memory usage, power consumption*$\}$ and $edgpu_j$ is an edge GPU, is a mapping function from cnn_i to $\mathbb{R}_+$. The function $\boldsymbol{T_{k,edgpu_j}}$ is defined below:

$$T_{k,edgpu_j} : cnn_i \rightarrow \hat{y_{ij}}$$
$$\hat{y_{ijk}} = T_{k,edgpu_j}(f_{i1}, f_{i2}, \ldots, f_{in})$$

Where $\hat{y_{ijk}}$ is the estimated value of the performance metric k of cnn_i on $edgpu_j$. In this paper, we study the case of three different NVIDIA edge GPUs, namely Jetson AGX Xavier, Jetson TX2 and Jetson Nano. Our methodology consists of applying the same modeling approach on each edge GPU. Given that we model each metric by a specific mapping function, we obtain three sets of prediction models for each edge GPU.

Unlike prior works on CNNs performances prediction such as [10, 31, 57, 64], which are based on a layer-wise performances estimation, our work aims to estimate CNN performances on a model-level granularity. To this end, we analyze the correlation between model-level features that describe the whole CNN architecture and the performance metrics. As demonstrated in the experimental results section 4, such a modeling approach leads to better generalization between different types of CNN architectures. The section below details the main CNN features considered in our proposed modeling methodology.

3.2 CNN characterization

Our proposed methodology characterizes CNN execution performances on a model-level granularity. Regarding the CNN architecture and the target edge GPUs, we assume that the following factors impact the performances of the CNN inference execution:

(1) Computational complexity, which directly impacts the GPU activities during the CNN inference;
(2) Memory workload, which corresponds to read and write memory operations for input, intermediate and output data;
(3) CNN hyperparameters, which corresponds to structural dependencies between computation and memory operations.

Considering the above factors, we search for the most correlated features with CNN performance. Since we target to model different performance metrics, the impact of the features may differ from one target performance metric to another. For instance, CNN features related to memory requirements are the most relevant to model CNN memory usage. We further discuss the CNN features related to each impact factor and the followed process to select the most relevant features for the prediction models.

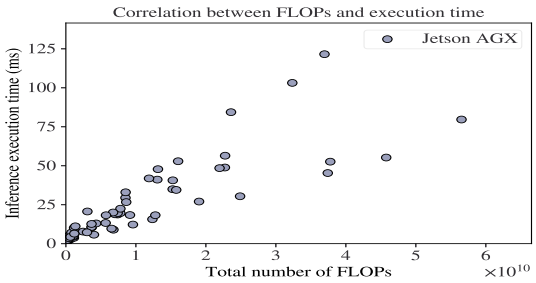

Fig. 3. Inference execution time VS. Flops for AGX

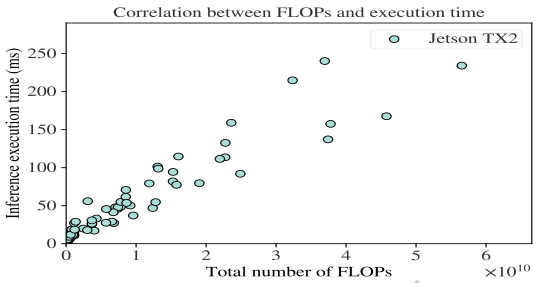

Fig. 4. Inference execution time VS. Flops for TX2

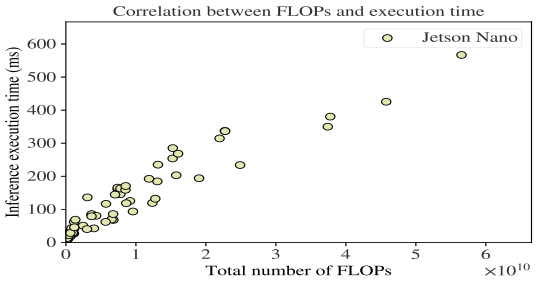

Fig. 5. Inference execution time VS. Flops for Nano

Table 2. FLOPs details for state-of-the-art CNNs

CNN	Conv2D	Add	Mul	Pool
ResNet-50	7.71B	31.02M	25.58M	1.81M
DenseNet-121	5.67B	7.89M	8.02M	1.98M
DPN-98	23.34B	70.54M	61.63M	2.71M
GoogleNet	3.00B	6.61M	6.64M	12.55M
ResNet-101V2	14.38B	52.32M	44.59M	2.16M
Inception-v3	5.67B	23.80M	23.85M	12.18M

3.2.1 Computational complexity. The theoretical total number of FLOPs (FLoating-point OPerations) is usually used to quantify the CNN computational complexity. This feature indicates the number of computations needed to perform a single CNN inference. As reported in table 2, this number is highly dominated by the total computations performed in convolutional layers, as they represent more than 98% of computations. This confirms that these layers are the bottleneck of computations in CNNs as they perform many Multiply-Add operations [36]. However, due to the optimization techniques used by GPUs during the CNN inference execution, the theoretical total number of FLOPs can not reflect precisely the performances. Figures 3, 4 and 5 show the relation between the total number of FLOP and the inference execution time for the 3 different HW platforms. Each point in the graphs corresponds to a CNN model from our benchmark. We can easily see from these figures that CNN inference execution time is not correlated to the number of FLOPs for a large set of models. Two models may have the same number of FLOPS but with different execution times. From figure 6.a we observe that there is no linear correlation between FLOPs and power consumption. Two CNN models may have the same number of FLOPS but consume very different power budget.

3.2.2 Memory requirements. For GPU platforms, memory activities have a significant impact on performances. Our experiments show that three factors can mainly explain the significant memory requirements of CNN inference:

(1) Reading CNN parameters, in terms of weights and biases,
(2) Reading the input data, writing the output results,
(3) Reading and writing the intermediate data, or activations, of the hidden layers.

Figure 6.c depicts the memory requirements for CNN parameters, activations and the measured system memory allocated for the DL framework -TensorFlow in our case- when running the CNN on the edge GPU.

We can notice that the allocated memory for the DL framework is significant and varies from one CNN to another. As mentioned in [41], existing DL frameworks such as TensorFlow, PyTorch and Caffe, do not take into account the unified memory of edge GPUs. Consequently, the data transfer between CPU and GPU creates redundant data copy on system memory. Moreover, to speed up the computation, the memory allocated to the DL framework cannot be released until the end of CNN inference, which requires a significant amount of memory for the DL framework. According to our experiments (figure 6.b) the memory usage of CNNs is strongly correlated with the sum of weights and activations. However, the high memory usage does not necessarily increase its execution time and power consumption. Indeed, data are not accessed with the same frequency during the inference. For instance, convolutional weights and activations are constantly accessed, whereas fully-connected weights are accessed once for each activation [56]. Moreover, execution time and power consumption may increase when activations and weights cannot be entirely loaded in the GPU cache memories. To quantify the impact of the number of memory accesses on execution time and power consumption, the CNN data-flow on the different levels of the memory hierarchy must be first determined. However, it is hard to extract memory activities without profiling the CNN on the target edge GPU. Therefore, this solution is not recommended as it is time-consuming and will add significant overhead to the prediction latency. To overcome this problem, we rely on CNN features that are mainly correlated to memory activities. Thus, we assume that both weights and activations highly impact memory activities when performing the inference. This assumption is verified experimentally.

3.2.3 CNN Hyperparameters. In addition to the previously mentioned features, we also include the architectural proprieties of CNN. We then consider the number of hidden layers, convolutional, fully-connected, batch normalization and pooling layers. Moreover, we study the impact of various image input sizes on CNNs. Varying the CNN image input size can be used along with the fine-tuning technique to adapt the CNN to new and/or customized training datasets.

We aim to generalize our performances prediction methodology to different use cases by varying the input image size. Indeed, we have observed that the execution time, power consumption and memory usage are strongly correlated to the CNN input image size. As shown in figure 7, increasing the input image size increases the execution time, power consumption and slightly the memory usage. To characterize the neurons within the CNN, we introduced a new feature, named *the weighted sum of neurons*. We calculate this metric by summing up the number of neurons in convolutional and fully-connected layers. The number of neurons in convolutional layers is then multiplied by the filter size: $height \times width \times depth$, to give more importance to neurons with large filter sizes. Nevertheless, The number of neurons in fully-connected layers is not weighted because the input neuron is associated with a single weight (scalar) and not a multidimensional filter.

3.3 Input features selection

We use the forward stepwise technique to select the most relevant CNN features for our prediction models. For each prediction model, we begin with an empty set of features. At each step, the most important feature is added where we use the F-score as a criterion of the feature's relevance. The F-score is calculated by the XGBoost [13] algorithm to rank the features from the most to the least important as shown in figure 8. The features are added to the prediction models until reaching a stopping criterion. We assess this latter by calculating the model prediction error using the Mean Absolute Percentage Error (MAPE). Hence, the stopping criterion is defined when no improvement can be observed in the prediction error after adding a new CNN feature. Feature's importance also depends on performances metrics. However, we have noticed a similarity in execution time and power consumption. Features related to the memory requirements are the most important for estimating memory usage.

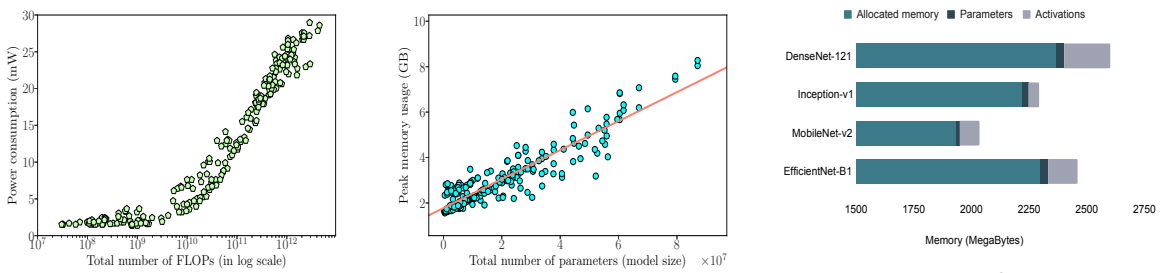

Fig. 6. a) On the left: correlation between power consumption and the total number of FLOPs (the two axes are represented in logarithmic scale) b) In the middle: correlation between the CNN memory usage and the total number of parameters. c) On the right: details about the memory usage of the CNN on the edge GPU

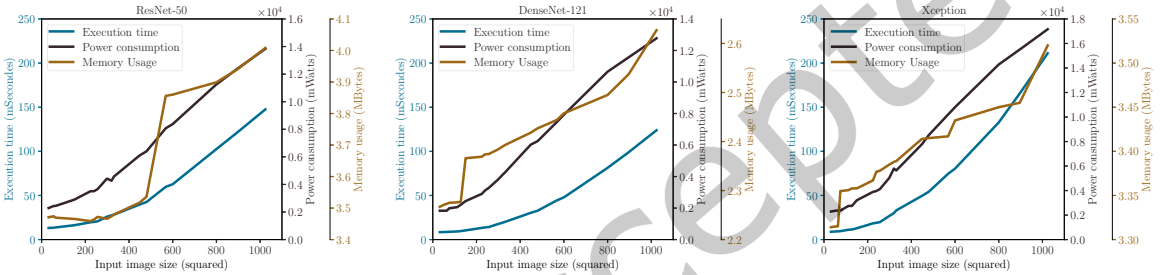

Fig. 7. Correlation between CNN image input size and the performance metrics: inference execution time, power consumption, memory usage. The reported values have been obtained by executing the three CNNs on the NVIDIA Jetson AGX Xavier.

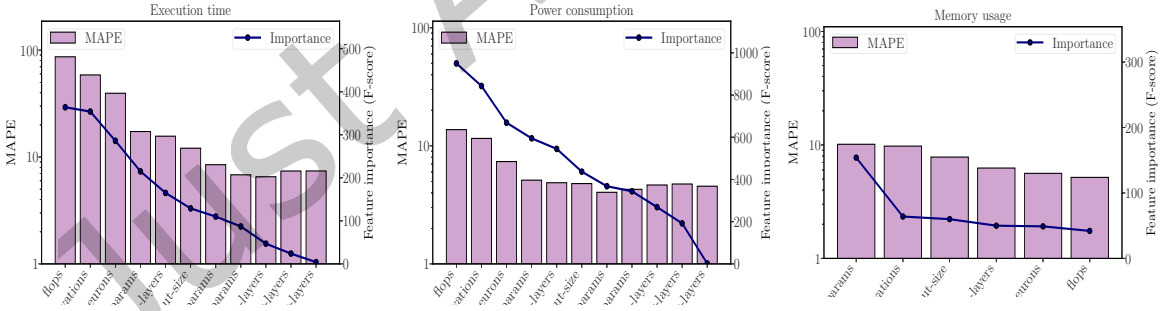

Fig. 8. The forward stepwise selection of features for CNN performances modeling on Jetson AGX Xavier. These figures show the results of the stepwise feature selection based on the calculated feature importance with the corresponding improvement in MAPE for each performance metric on the Jetson AGX Xavier. Features are ranked and added to the prediction model based on their relevance (i.e., features with high F-score metrics are the most relevant). On the left axis: the feature importance calculated by XGBoost. On the right axis: improvement of the Mean Absolute Percentage Error (MAPE). A similar trend has also been observed for the two other edge GPUs (Jetson TX2 and Nano).

3.4 Prediction algorithms

3.4.1 Ridge Polynomial Regression (Poly). Polynomial regression (Poly) [49] is a particular form of linear regression where we search to fit a polynomial equation on the data with a curvilinear relationship between the input features and output variables. The polynomial regression is formulated as follows:

$$\hat{y_{ij}} = \alpha_p X_p^d + \alpha_{p-1} X_{p-1}^{d-1} ... + \alpha_2 X_2^2 + \alpha_1 X_1 + \beta + \epsilon \tag{1}$$

Where $(X_1^1, \ldots, X_p^d)$ refers to the vector of polynomial features created from the input features vector of cnn_i, whereas d indicates the polynomial degree. Polynomial regression suffers from overfitting, especially for high values of polynomial degree. To overcome this problem, we rely on the Ridge regularization technique [23] that penalizes the polynomial features that could present multicollinearity by minimizing the following loss function:

$$\Phi(y_{ij}, \hat{y_{ij}}) = \min \sum_{i=1}^{i=n} (y_{ij} - \hat{y_{ij}})^2 + \lambda \sum_{k=1}^{k=p} w_k^2 \tag{2}$$

We tune both the polynomial degree d and the regularization factor λ as reported in table 15 in the appendix.

3.4.2 Support Vector Regression (SVR). Support Vector Regression (SVR) [6], which is based on Support Vector Machine (SVM) [22], has been proved to be effective for regression problems. SVR deals with non-linearity by using kernels that map the input feature from the original space to a higher-dimensional space. This non-linear transformation helps find the optimal hyper-plane by expanding the original dataset's dimensionality so that the features can be linearly separable. In general, the hyperplane found by the SVR can be formulated as follows:

$$f(X) = wX + b \tag{3}$$

where w is the weight associated with each support vector X for its importance to the prediction. The most important hyperparameters to tune for the SVR are reported in table 15.

3.4.3 Multi-Layer Perceptrons (MLP). The Multi-Layer Perceptrons (MLP) [45] is a part of Artificial Neural Networks (ANN). The architecture of the MLP can indeed model non-linear relationships by using activation functions which are non-linear transformations. The MLP is a succession of hidden layers where each layer applies the following transformation on the input features:

$$\hat{z} = \theta(\sum_{i=1}^{n} w_i x_i + b) \tag{4}$$

where $\hat{z}$ is the output value, w_i is the weight of the feature x_i, b is the bias and θ is the activation function. We note here that x_i can refer to one of the input features of cnn_i -the case of the first layer of the MLP- or extracted features from a resulting output vector of a hidden layer. MLP networks require many training data to provide the best generalization and prevent overfitting. We design the optimal MLP models by using the grid search on the set of hyperparameters defined in table 15.

3.4.4 Random Forest (RF). Random Forest (RF) [37] is one of the ensemble learning methods which is based on the bagging technique where predictions are made by multiple predictors (i.e., decision trees). Each of them is trained on a subset of training samples and a subset of features selected randomly. The final prediction is calculated by averaging the predictors' outputs. Technically, this method reduces the error variances since multiple decision trees contribute to the final prediction. The most important hyperparameters to tune for the RF are summarized in table 15.

3.4.5 eXtreme Gradient Boosting (XGBoost). eXtreme Gradient Boosting (XGBoost) [13] is an ensemble learning-based algorithm. Unlike Random Forest, XGBoost is based on the boosting technique that makes predictions from weak predictors arranged sequentially: The first predictor is trained on the entire dataset, where the subsequent predictors are trained on the residuals of prior predictors. This technique helps to focus on improving the mispredicted values. XGBoost creates weak predictors using the gradient descent algorithm until achieving an acceptable prediction error. XGBoost has several hyperparameters to tune, which are summarized in table 15.

This section detailed our proposed approach according to three perspectives: First, we formulated the problem statement of predicting CNN performances on edge GPUs. Then, we introduced the considered CNN features and gave details about their impact on each performance metric. Finally, we gave an overview of the used ML algorithms for the modeling step. In the next section, we present the evaluation step used to assess the effectiveness of our proposed approach.

4 EVALUATION METHODOLOGY

This section details the followed evaluation methodology and the obtained results. As shown in figure 2, our modeling process is subdivided into three main steps: 1) Benchmarking, 2) Data collection, and 3) Modeling. These three steps are performed in a pipeline fashion. In this section, we first detail each step of this pipeline. Then, we give details about the experimental setup. Finally, we present and discuss our obtained results.

4.1 CNN Benchmarking

The benchmarking step mainly consists of deploying and executing the CNNs inference the target edge GPUs. The benchmarks have been designed based on state-of-the-art CNNs for image classification and executed on different edge GPUs. Figure 9 presents a taxonomy of the CNN architectures that have been investigated in this work. As depicted in figure 9, we have considered different architectures widely used in the literature to build CNN models. From depth and width-based CNN such as ResNet and Inception to multi-path and multi-connection-based CNN such as ShuffleNet and MobileNet. Hence, our dataset covers a wide range of different CNN architectures. In our work, we do not have to train the studied CNN models. Instead, we randomly set their weights values. This is because the CNN weights values have no impact on the inference performances regarding execution time, power consumption, and memory usage. However, they directly impact the CNN accuracy. A 32-bits floating representation has been fixed for the weights and tensors representation. Finally, in order to expand our datasets to characterize different types of correlations, we apply data augmentation techniques by varying three factors:

(1) Input Image Size: The impact of the input image size on the CNN performances is studied. For this purpose, we test the frequently used image resolutions, from 32*32 to 2400*2400 pixels, with three channels for the RGB representation.
(2) CNN Variants: Different variants of the same CNN architecture are considered. For instance, we deploy different variants of the ResNet architecture [20, 21] by varying the number of layers and residual blocks. Hence, we obtained eight (8) widely used variants of ResNet, such as ResNet18, ResNet34, ResNet50, and ResNet101.
(3) CNN Architectures: Finally, we consider different CNN architectures to quantify their impact on inference performances.

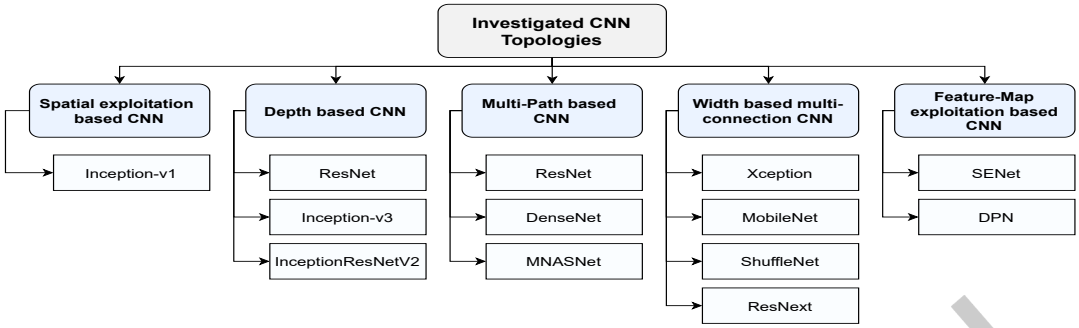

Fig. 9. Taxonomy of the investigated CNN architectures based on the classification given by [32]

Table 3. Details of the CNN benchmarks used in the experiments. The first line of the table gives the used CNN architectures. The second line gives the number of the considered variants for each CNN architecture. The third and fourth lines report the number of the considered input sizes and their intervals of values. In total, we have 2056, 1975, 1612 input-data for Jetson AGX, TX2, and Nano, respectively: 70% of data have been used for training while 30% of data for test and prediction error evaluation

CNN	GoogleNet	Inception (v3)	InceptionResNet (v2)	DPN	DenseNet	Xception	EfficientNet	MNASNet	ResNet (v1)	ResNet (v2)	MobileNet (v1)	MobileNet (v2)	MobileNet (v3)	ResNext	SENet	ShuffleNet (v1)	ShuffleNet (v2)
# variants	1	1	1	4	5	1	4	5	12	7	4	5	4	2	6	4	3
# input image sizes	15	20		23			25		27								
Interval of image sizes (from.. to..)	(224x224x3), (1024x1024x3)]	(75x75x3), (1024x1024x3)]		(32x32x3), (1024x1024x3)]			(32x32x3), (1600x1600x3)]		(32x32x3), (2400x2400x3)]								

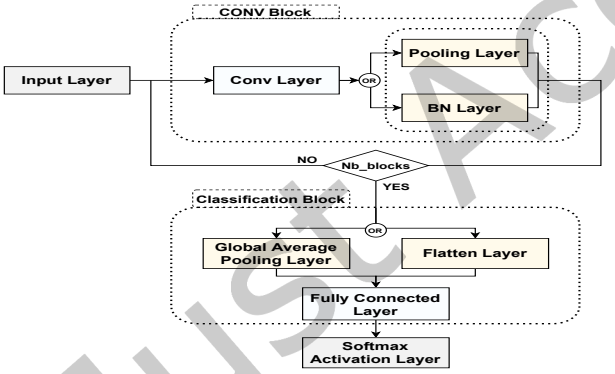

Fig. 10. CNN baseline for NCA

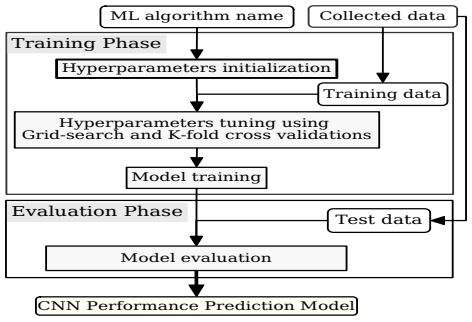

Fig. 11. Prediction model design

4.2 Data Collection

This step consists of collecting all the data needed to characterize and model the CNN performances. It involves the extraction of four (4) sets of measurements. The first set of data comprises the CNN features used as prediction variables. This type of data is the easiest to extract as it does not require a further deployment of the CNN on the edge GPU. Nevertheless, the other three sets are composed of the real measurements of the execution performance metrics (i.e., execution time, power consumption, and memory usage). Consequently, they can not be obtained without consistent profiling of the CNN inference execution on the target edge GPU. Figure 12

depicts the workflow of the CNN profiling process to collect the performances measurements. Below we give more details about the data collection step:

- *CNN features extraction* : We have developed a parser that takes the CNN architecture description as input and gives its essential features in output. We give more details about the extracted features and their ranges in table 4. From this table, we can notice that a wide range of values has been considered for each CNN feature. Thus, the studied CNNs range from small to large models according to the reported values of FLOPs, parameters, activations, and the number of layers in table 4. Hence, our benchmark covers different types and values of CNN input features.
- *Execution time profiling* : CNN inference execution time measurements have been collected using the Nvidia profiling tool Nvprof [47]. We measure the CNN execution time by summing up the execution times of the GPU kernels that have been invoked during the CNN inference. To minimize the impact of the profiling overhead on the measurements, we repeat each experiment 100 times on 100 randomly chosen images. We then consider the average measured execution times over the 100 experiments according to the recommendation given in [46]. We have also noticed that the measured execution times of the replicated experiments are normally distributed; therefore, we consider the mean as a central tendency.
- *Power consumption profiling* : To measure the power consumption of the CNN inference, we use the onboard GPU sensors of NVIDIA Jetson platforms. These sensors can be read automatically and periodically with the tegrastats command utility [48]. When running CNN inference, We also run a background process to periodically examine the GPU power consumption, with a sampling period set to the minimum value (i.e., 1 ms). To ensure the reliability of the power consumption measurements, we disconnect all the peripheries connected to the GPU board during measurements except the port for SSH communication. We repeat the experiments 100 times for Jetson AGX Xavier as the inference times are generally short and 50 times for Jetson Nano and TX2. We have noticed that measured power consumption over the repeated experiments form a skewed normal distribution as power consumption tends to be relatively high in the first runs because of the GPU warm-up. Hence, we consider the median of the peak power consumption of the replicated experiments as a central tendency.
- *Memory Usage Profiling* : NVIDIA Jeston Platforms are characterized by their unified memory. Thus, the GPU and the CPU share the same physical memory. In this case, the GPU memory usage is not limited, and it can use all of the available system memory. This particularity of Nvidia Jetson platforms makes it possible to monitor the GPU memory usage by monitoring the system memory. For this reason, we measure the peak memory usage of the CNN with the tegrastats utility [48] by monitoring the system memory usage in the same way as for power consumption. We define the CNN memory usage as the peak system memory usage during the CNN inference minus the initial memory usage before running the CNN inference.

In addition, we have adjusted the collected data for each edge GPU as follows:

- We consider only CNN models with an execution time higher than 1ms to meet the sampling interval constraint of the edge GPU performance profiler (i.e., tegrastats).
- We excluded CNN models that don't fit in the edge GPU due to their memory requirements.

Hence, the size of the smallest CNN executed by the three edge GPUs is 0.27M (in terms of number of parameters). However, the size of the largest CNN depend on the edge GPU's memory capacity. We have observed that the Jetson AGX Xavier can execute CNNs with size up to 125M, whereas Jetson TX2 and Nano can execute CNN with sizes up to 87M and 50M, respectively.

4.3 Model Design

The resulting datasets from the benchmarking phase can be denoted as $D_{jk} = \{cnni; y_{ijk}\}_{i=1}^{n}$, for each performance metric denoted k where $k \in$ {execution time, power consumption, memory usage} and each edge GPU denoted

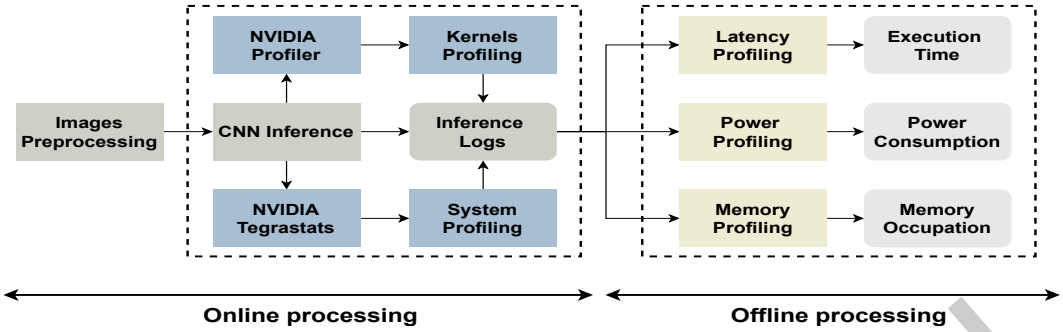

Fig. 12. CNN performances profiling workflow

$edgpu_j$. Once the datasets D_{jk} are obtained, we train sets of prediction models for each performance metric k and edge GPU $edgpu_j$ on the corresponding dataset D_{jk}. At this step, each prediction model learns the statistical correlations between the features vectors cnn_i and the predicted variable y_{ijk}, which corresponds to the measured values of one of the performance metrics k. To perform the learning phase on the collected data D_{jk}, we rely on different Machine Learning algorithms. Our process to design the prediction models is depicted in figure 11, and it has two main inputs:

(1) *ML algorithm name*, which corresponds to one of the ML algorithms described in section 3.4.
(2) *Collected data*, which are the datasets D_{jk} obtained from the benchmarking and data collection steps. Table 3 gives more details about our CNN benchmarks. Table 4 provides more details about the extracted data about the CNN architectural features from the benchmarks detailed in table 3.

The data pre-processing step includes two important points:

- *Dimensionality reduction:* To prevent the risk of overfitting, we perform a feature selection on the prediction variables (i.e., CNN features detailed in section 3.2). We only consider the most relevant features for each prediction model strongly correlated with the target performance metric.
- *Data transformation:* We have noticed non-trivial correlations between CNN features and power consumption from our correlation analysis study. As pointed by [10], power consumption has inherent limitations, i.e., it can only take a limited set of values that are restricted to the power budget fixed by the hardware constructor. Thus, power consumption does not increase linearly as the CNN computational complexity and/or memory requirements increase. Consequently, to model this correlation tendency, we apply a logarithmic transformation to the prediction input features as suggested by [10]. Hence, in this case, the input vector cnn_i contains the logarithmic terms of the prediction input features instead of the real values. We must note that this transformation is only applied for the case of power consumption.

To train, validate and test our prediction models, the collected data have been split into two sets: 70% of data have been dedicated for training and cross-validation and 30% of data have been reserved for prediction models test and evaluation.

After the data pre-processing step, we construct the predictions models according to the process represented in figure 11. We first perform the hyperparameters tuning of the prediction models. To do so, the hyperparameters' search space is first initialized. Afterward, The tuning is realized using the grid search and K-fold cross-validation techniques to select hyperparameters' best values. Second, once the optimal combination of hyperparameters is found, we train the prediction models on the 70% data for training. Finally, we test the final prediction models on the 30% data for the test. We have considered several CNN configurations and architectures for the test phase to

evaluate the prediction error and the generalization power of the obtained prediction models towards different performance metrics and edge GPUs. To this end, according to the considered configurations of our datasets (i.e., CNN architectures, variants, and input size), we have categorized the test data according to their complexity into the following three exploration spaces:

(1) *Performance estimation of New Image Sizes (NIS):* We evaluate the prediction models on state-of-the-art CNNs with new input image sizes in this first exploration space. We vary the input size from 32*32 to 2400*2400 pixels, depending on the CNN architecture as detailed in table 3. That is, for each CNN architecture we test different input sizes. For instance, for GoogleNet [59], we vary the input size from 224x224x3 to 1024x1024x3, whereas for DenseNet [27], we vary the input size, for each of its five (5) variants, from 32x32x3 to 1024x1024x3 By varying the input size of the CNN, we obtain different values of FLOPs, intermediate activations, neurons, and eventually different numbers of parameters. However, the number and type of layers remain the same. Therefore, this is the simplest exploration space as we only vary the input size of already seen CNN architectures in the training dataset.

(2) *Performance estimation of New CNN Variants (NCV):* We evaluate the prediction models when new CNN variants are considered in this second exploration space. Based on state-of-the-art CNN architecture templates, we derived new CNNs by varying scaling factors that control the CNN model's width, depth, input size, and operators. These factors are, for instance, the depth and width multipliers for MobileNet [25] and EfficientNet [61], the number of layers and residual blocks for ResNet [20, 21], the number of output channels for ShuffleNet [71, 71], and the depth, reduction ratio, and the number of residual blocks for SENet [26]. If we consider ResNetV1 [20] as an example, we trained our predictors on ResNetV1-18 to ResnetV1-101, and we predict the execution time, power consumption, and memory usage for ResNetV1-152 with new numbers of FLOPs, activations, neurons, parameters, and layers. Hence, this exploration space is more complicated than the first one as we vary different scaling factors of already seen CNN architectures in the training dataset.

(3) *Performance estimation of New CNN Architectures (NCA):* We evaluate our prediction models on new synthetic CNN architectures in this exploration space. Figure 10 represents the CNN baseline architecture used to construct the NCA exploration space. To generate synthetic CNNs from this baseline, we randomly vary the input size and convolutional blocks configurations by varying numbers and types of convolutional layers, pooling, and batch normalization layers. We also randomly vary the configurations for each layer, for instance, the number and size of kernels in convolutional layers, stride and kernel size in pooling layers, and the number of units in fully-connected layers. Unlike NIS and NCV exploration spaces, the CNN architectures for NCA are generated from scratch and are not based on existing stated-of-the-art CNN architecture templates. This exploration space is hence the most complicated, compared to the two first ones, as we evaluate the capacity of the five models to predict the performances of completely new CNN architectures not included in the training dataset.

Table 4. Details of the CNN features

CNN feature	Range
# of hidden layers	[10 – 4072]
# of CONV layers	[4 – 1825]
# of BN layers	[0 – 265]
# of FC layers	[0 – 18]
# of filters per CONV layer	[3 – 2688]
# of units per FC layer	[0 – 5625x10^3]
Filter size	[(1x1) – (11x11)]
Input size	[32 – 2400]
# of CONV layers parameters	[0.27x10^6 – 87.1x10^6]
# of BN layers parameters	[0 – 0.8x10^6]
# of FC layers parameters	[0 – 119.8x10^6]
Total number of FLOPs	[3.1x10^6 – 5.2x10^{12}]
Sum of intermediate activations	[0.1x10^6 - 37.6x10^9]

Table 5. Edge GPUs used for experiments

Hardware feature	Jetson Nano	Jetson TX2	Jetson AGX
CPU	4-core A57 1.43 GHz	6-core Denver + A57 2 GHz	8-core Carmel 2.26 GHz
Memory capacity	4 GB 64-bit LPDDR4	8 GB 128-bit LPDDR4	16 GB 256-bit LPDDR4x
Memory bandwidth	25.6 GB/s	58.4 GB/s	137 GB/s
GPU	128-core Maxwell 1.23 GHz	256-core Pascal 1.3 GHz	512-core Volta 1.37 GHz
Power mode	5W/10W	7.5W/15W	10W/15W/30W

4.4 Experimental Setup

To characterize the CNN inference performances, we deploy several CNNs on different edge GPUs. We use three Nvidia GPU platforms from the Jetson series dedicated to edge computing: Jetson Nano, Jetson TX2, and Jetson AGX Xavier. The hardware specifications for each platform are described in table 5. We have configured these platforms to profiling mode (MAXN power mode). We have used the same underlying software configuration for the three edge GPUs. Table 5 shows that these platforms are very different in terms of computing/memory capacities and CPU/GPU micro-architectures. For instance, Jetson AGX Xavier offers excellent computation speed and memory capacity at the cost of high power consumption. In comparison, Jetson Nano provides reduced power consumption but lower computation speed and memory capacity. Jetson TX2 delivers good performances between Jetson AGX Xavier and Jetson Nano.

CNNs have been implemented on the edge GPUs using the Keras 2.3.1 API with TensorFlow 1.14 as backend [1]. This framework is running on top of Cuda version of 10.0. The host operating system in both platforms is Linux Ubuntu 18.04.3 LTS with a kernel 4.9.149-Tegra. We focus in this paper on evaluating CNN inference with TensorFlow on top of CuDNN API [15]. However, we believe that our prediction methodology can also be applied when considering another high-performance Deep Learning SDK such as TensorRT [62] which is composed of optimized DL routines built with the CuDNN API. This is justified by the fact that our prediction methodology relies on characterizing the CNN on model-level granularity while abstracting details of the deployment environment, such as the used software configuration.

5 APPROACH ANALYSIS

5.1 Results Discussion

This section provides a complete and comprehensive evaluation and analysis of our proposed prediction methodology. The obtained results are assessed for each performance metric: execution time, power consumption, and required memory usage to run the CNN inference. Three edge GPUs are considered: Jetson Nano, TX2, and AGX Xavier. We evaluate our modeling methodology from three perspectives:

- First, we evaluate the prediction error of each prediction model using the Mean Absolute Percentage Error (MAPE) [16]. To calculate the MAPE between the predicted ($\hat{y_{ij}}$) and measured (y_{ij}) values of performance metrics, we use the mathematical formula given by equation 5. As we have different ranges and values scales for execution time, power consumption, and memory usage, we choose the MAPE as a scale-independent evaluation metric to easily interpret the obtained results.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_{ij} - \hat{y_{ij}}}{y_{ij}}\right| \quad (5)$$

- Second, we compare the prediction models according to the time needed for: training, tuning, and exploring the values of their corresponding hyperparameters and finally, for performing the prediction on the input features vector (cnn_i).
- Third, we discuss the correlation coefficients between the measured and predicted performance metrics. To this end, We use both Kendall-tau [3], and Pearson [7] rank correlation coefficients to know whether the prediction models respect the rank of the estimations or not regarding the rank of the measured values of performance metrics.

5.2 Execution Time Prediction

Figure 13 gives the obtained Mean Absolute Percentage Error (MAPE) and their corresponding confidence interval of 95%. Table 6 summarizes the analysis of the five (05) prediction models in terms of MAPE, training, tuning costs, and prediction latency. Figure 7 reports the calculated rank correlation coefficients of all the obtained results for CNN execution time estimations.

5.2.1 ***Prediction error and generalization power:*** From figure 13 and Table 6, we notice that the MAPE values are ranging from ∼ 7% and ∼ 16%. We can also notice that the lowest MAPE values are generally obtained for NIS and NCV, whereas the highest MAPE values are always obtained for NCA. This finding corroborates the hypothesis on the complexity of NCA compared to NIS and NCV, as we have already detailed in section 4.

For the three exploration spaces (i.e., NIS, NCV, and NCA), XGBoost and Ridge Polynomial Regression outperform the other prediction models and offer the lowest MAPE values. On the one hand, XGBoost is among the most powerful ML algorithms; the Boosting technique to reduce the prediction error through many estimators arranged sequentially makes it very accurate. On the other hand, polynomial regression offers both flexibility and low prediction error. It also does not require a lot of data for training. These proprieties make it widely used for systems and performance modeling. We implement the Ridge regularization technique to penalize the polynomial terms with lower contributions in the prediction to prevent overfitting.

Table 6. Execution time prediction models analysis: prediction error, training, tuning, and latency

Prediction Model	Test Data	MAPE			Training Time	Tuning Time	Prediction Latency		
		Nano	TX2	AGX			AGX	TX2	Nano
Poly	NIS	10.16%	10.29%	9.55%	**12.72 ms**	**6.32 mn**	**357.9 ns**	**528.1 ns**	**704.3 ns**
	NCV	11.55%	11.06%	10.20%					
	NCA	11.54%	13.19%	13.04%					
MLP	NIS	10.98%	12.47%	12.17%	1.13 s	11.09 hr	22.7 us	30.9 us	53.8 us
	NCV	12.37%	9.97%	13.80%					
	NCA	13.65%	15.18%	13.18%					
SVR	NIS	11.83%	14.97%	14.68%	159 ms	22.6 hr	41.8 us	52.1 us	78.5 us
	NCV	**9.29%**	**9.30%**	**7.92%**					
	NCA	16.01%	16.86%	15.86%					
RF	NIS	9.39%	10.97%	11.94%	3.5 s	4.6 hr	1.1 ms	1.5 ms	2.4 ms
	NCV	**9.14%**	**8.91%**	**10.41%**					
	NCA	13.97%	13.79%	13.76%					
XGBoost	NIS	**9.12%**	**9.58%**	**8.28%**	**538.5 ms**	**13.22 mn**	**2.1 us**	**3.5 us**	**6.2 us**
	NCV	**7.99%**	**9.09%**	**7.45%**					
	NCA	**13.80%**	**13.11%**	**11.74%**					

RF depicts comparable performance to XGBoost and Polynomial regression. RF is composed of different decision trees trained on random subsets of training data samples and features. This property helps to reduce the

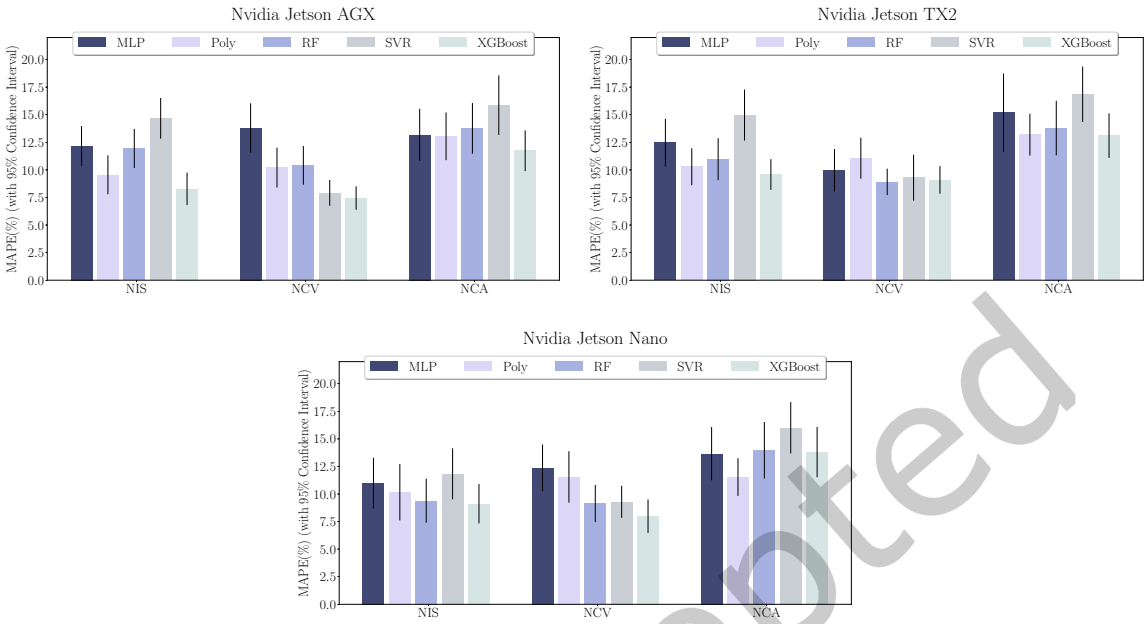

Fig. 13. Mean Absolute Percentage Error (MAPE) for execution time on the three edge GPUs: Nvidia AGX, TX2, and Nano, respectively, with the corresponding 95% Confidence Interval. In this figure, we compare the five (05) prediction models for the three exploration spaces: New Image Sizes (NIS), New CNN Variants (NCV), and New CNN Architectures (NCA).

Table 7. Rank correlation coefficients between Measured and predicted execution times for the three exploration spaces: NIS, NCV, and NCA, on different NVIDIA edge GPUs: Nano, TX2, and AGX

Prediction model	Test Data	Kendall coefficients			Pearson coefficients		
		Nano	TX2	AGX	Nano	TX2	AGX
Poly	NIS	0.949	0.935	0.93	0.998	0.998	0.997
	NCV	0.938	0.935	0.93	0.984	0.998	0.997
	NCA	0.961	0.935	0.93	0.995	0.998	0.997
MLP	NIS	0.937	0.936	0.932	0.998	0.994	0.996
	NCV	0.935	0.936	0.932	0.982	0.994	0.996
	NCA	0.955	0.936	0.932	0.992	0.994	0.996
SVR	NIS	0.932	0.924	0.93	0.995	0.989	0.994
	NCV	0.94	0.924	0.93	0.982	0.989	0.994
	NCA	0.949	0.924	0.93	0.993	0.989	0.994
RF	NIS	0.95	0.935	0.933	0.997	0.994	0.997
	NCV	0.945	0.935	0.933	0.979	0.994	0.997
	NCA	0.945	0.935	0.933	0.991	0.994	0.997
XGBoost	NIS	**0.95**	**0.948**	**0.942**	**0.994**	**0.995**	**0.997**
	NCV	**0.953**	**0.948**	**0.942**	**0.979**	**0.995**	**0.997**
	NCA	**0.945**	**0.948**	**0.942**	**0.982**	**0.995**	**0.997**

variance error. MLP generally has good performances with a slight loss of generalization for NIS and NCA. This nonperformance can be due to its nature which tends to overfit data. In addition, as detailed in the next section, MLP is very sensitive to the variation of hyperparameters, making it very hard to tune. SVR is less accurate than the models mentioned above for NIS and NCA. The variation in the MAPE values in the three exploration spaces is relatively high, interpreted as overfitting.

If we compare the obtained results of the three edge GPUs: Jetson Nano, TX2, and AGX, we notice that the prediction errors regarding the three (3) exploration spaces are very similar. Hence, our modeling approach is easily portable to other platforms to obtain CNNs inference execution time predictions.

5.2.2 ***Training and tuning costs:*** Tuning and training costs differ from one model to another. However, we have noticed that most prediction models are sensitive to hyperparameter configuration and need time and effort to tune them well. For instance, XGBoost shows sensitivity to hyperparameters, precisely, to the Booster hyperparameters that need to be set carefully to achieve the best performances. Nevertheless, the training and tuning times are relatively short for XGBoost, achieving the best compromise between prediction error and modeling cost.

Prediction approaches based on ensemble learning, such as RF and XGBoost, need to be trained on different data scales to identify an effective mapping between input features and inference execution times. According to our experiments, the prediction error converges by increasing the number or depth of the decision trees in Random Forest. These two (2) factors increase the complexity of the training and the prediction latency in RF. For this reason, when a short interval of time for tuning and training is desired, XGBoost will be more efficient than RF.

Ridge Polynomial regression (Poly) has two hyperparameters to tune: The polynomial degree and the regularization factor, and the two of them considerably impact the prediction error. The optimal value of the polynomial degree depends on the scales of execution times which vary from one edge GPU to another. MLP is very sensitive to the variation of the hyperparameters, making it very hard to tune. In fact, the size of the MLP has an important impact on the prediction error. We have noticed that large MLP networks are prone to overfitting compared to small ones. SVR is sensitive to the type of kernels and the cost (C) (see section 3.4.2). According to our results, linear kernels perform the best for execution time modeling, whereas the small C values, lower than 1, lead to a considerable loss of generalization. From table 6, we notice that SVR has a drawback of presenting a considerable tuning time. Different kernels have been tested. Polynomial and RBF kernels take the longest time for training, increasing the tuning time of SVR.

5.2.3 ***Rank-preserving:*** We analyze the rank correlation between the measured and predicted inference execution times to determine whether the prediction models are rank-preserving or not. Preserving the ranking of CNN architectures regarding their predicted performances on the different edge GPUs is very important during the design space exploration. For this purpose, we used both Kendall-tau and Pearson correlation coefficients. These correlation coefficients evaluate the nature and the degree of similarity between the two (2) sets of data: the set of measured and predicted inference execution times. Figure 7 gives the details about the obtained rank correlation coefficients.

From figure 7, we observe that all of the prediction models highly preserve the rank as their Kendall coefficients range from 0.94 to 0.97 and their Pearson coefficients range from 0.98 to 0.99. We can, therefore, conclude that our prediction methodology can be used to rank the CNN according to their estimated execution times. Hence, The obtained ranking can be used to select the best CNN to be deployed on the edge GPU.

5.3 Power Consumption Prediction

Figure 14 gives the obtained MAPE values with their corresponding confidence interval of 95%. Table 8 summarizes the analysis of the five (05) prediction models for CNN power consumption in terms of MAPE, training, tuning costs, and prediction latency. Figure 9 reports the calculated rank correlation coefficients of all the obtained results for CNN power consumption estimations.

5.3.1 ***Prediction error and generalization:*** Similar to the case of inference execution time, we have two main observations: First, the lowest prediction errors are obtained for NIS and NCV, while the highest error is

Table 8. Power consumption prediction models analysis: prediction error, training, tuning, and latency

Prediction Model	Test Data	MAPE			Training Time	Tuning Time	Prediction Latency		
		Nano	TX2	AGX			AGX	TX2	Nano
Poly	NIS	6.18%	7.02%	5.85%	**11.29 ms**	**6.09 mn**	**401.1 ns**	**713.9 ns**	**900.6 ns**
	NCV	6.51%	6.97%	5.66%					
	NCA	7.10%	7.05%	5.71%					
MLP	NIS	5.66%	6.72%	5.72%	1.9 s	15.33 hr	23.2 us	27.4 us	39.0 us
	NCV	7.96%	7.41%	6.43%					
	NCA	7.22%	7.19%	6.03%					
SVR	NIS	**4.51%**	**5.14%**	**3.94%**	771.3 ms	25.1 hr	201.3 us	511.2 us	803.5 us
	NCV	5.39%	6.50%	5.33%					
	NCA	8.29%	8.57%	7.81%					
RF	NIS	4.96%	5.82%	5.88%	3.31 s	4.6 hr	908.5 us	1.2 ms	1.7 ms
	NCV	6.94%	5.84%	4.13%					
	NCA	5.18%	6.75%	5.81%					
XGBoost	NIS	**3.56%**	**5.24%**	**4.51%**	**488.5 ms**	**10.12 mn**	**4.7 us**	**6.0 us**	**8.9 us**
	NCV	**4.91%**	**5.20%**	**5.69%**					
	NCA	**4.62%**	**6.04%**	**5.31%**					

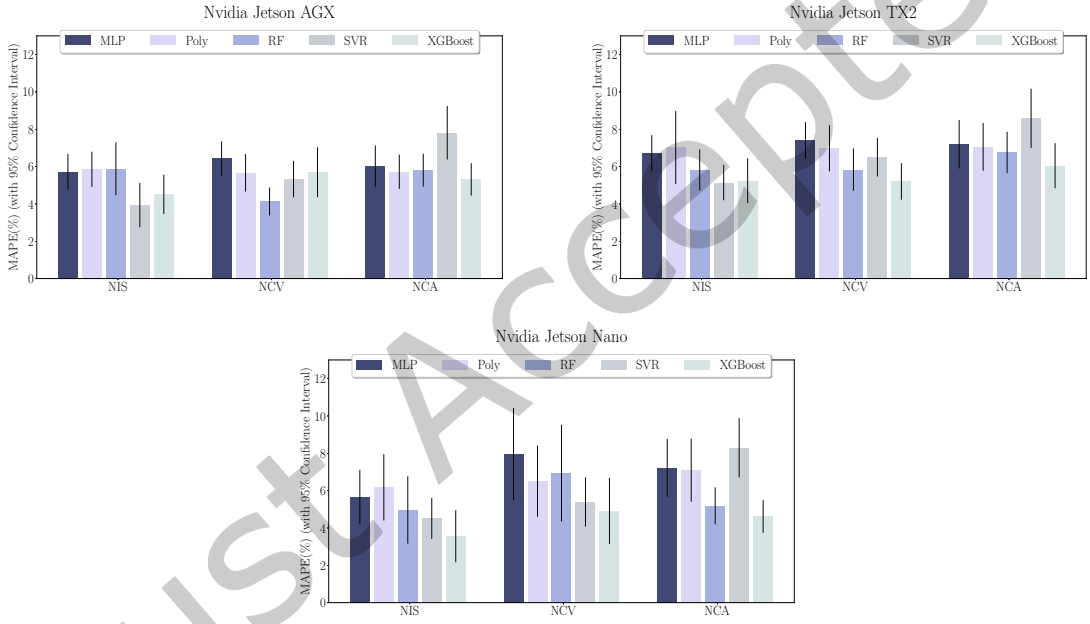

Fig. 14. Mean Absolute Percentage Error (MAPE) for power consumption on the three NVIDIA edge GPUs: AGX, TX2, and Nano, respectively, with the corresponding 95% Confidence Interval. In this figure, we compare the five (05) prediction models for the three exploration spaces: New Image Sizes (NIS), New CNN Variants (NCV), and New CNN Architectures (NCA).

obtained for NCA. This is due to the complexity of the exploration spaces, as we have explained in sub-section 5.2.1. Second, both XGBoost and Ridge polynomial regression provide the lowest prediction error for power consumption estimation on the three edge GPUs. These two models also achieve a good generalization between the three exploration spaces. From figure 14 and table 8, we can also notice that the MAPE values are generally between ~ 3% and ~ 8%, which indicates both reduced bias and variance errors. This figure also shows that the obtained MAPE for power consumption is smaller than the obtained MAPE for execution time. The main reason

is that edge GPU hardware architectures are simple, and their power consumption does not vary considerably during the inference.

RF and MLP depict similar performances in terms of prediction error and generalization. SVR provides low prediction error for NIS and NCV but a quiet loss of generalization for NCA. This same tendency was observed for execution time prediction. These observations indicate that SVR tends to overfit data, performing well for known CNN architectures and variants but losing generalization for unseen CNN architectures (NCA).

Table 9. Rank correlation coefficients between Measured and predicted values of power consumption for the three exploration spaces: NIS, NCV and NCA, on different NVIDIA edge GPUs: Nano, TX2 and AGX

Prediction model	Test data	Kendall coefficients			Pearson coefficients		
		Nano	TX2	AGX	Nano	TX2	AGX
Poly	NIS	0.769	0.926	0.94	0.952	0.987	0.995
	NCV	0.911	0.926	0.94	0.985	0.987	0.995
	NCA	0.657	0.926	0.94	0.935	0.987	0.995
MLP	NIS	0.758	0.942	0.931	0.977	0.994	0.994
	NCV	0.88	0.942	0.931	0.976	0.994	0.994
	NCA	0.74	0.942	0.931	0.966	0.994	0.994
SVR	NIS	0.765	0.952	0.957	0.974	0.996	0.996
	NCV	0.899	0.952	0.957	0.983	0.996	0.996
	NCA	0.542	0.952	0.957	0.896	0.996	0.996
RF	NIS	0.836	0.935	0.934	0.976	0.992	0.992
	NCV	0.9	0.935	0.934	0.98	0.992	0.992
	NCA	0.743	0.935	0.934	0.97	0.992	0.992
XGBoost	NIS	**0.905**	**0.945**	**0.947**	**0.979**	**0.995**	**0.996**
	NCV	**0.934**	**0.945**	**0.947**	**0.991**	**0.995**	**0.996**
	NCA	**0.737**	**0.945**	**0.947**	**0.975**	**0.995**	**0.996**

As observed for execution time prediction, the three edge GPUs' prediction error and generalization tendency are quite similar. However, the reported MAPE values for the Jetson Nano are relatively smaller than for other platforms. On the one hand, this is due to the available power budget of the Jetson Nano, which is limited to 5 Watts in our case. On the other hand, Most of the studied CNNs are computationally and memory intensive, which results in the highest possible power consumption for most cases. Thus, obtained power consumption values for these CNNs are quite similar.

5.3.2 ***Training and tuning costs:*** From table 8, we can observe that the training and tuning costs of the prediction models for power consumption have similar tendencies as the execution time. This can be explained by the fact that we are using the same modeling methodology and data for each combination of performance metric and edge GPU. Nonetheless, we have also noticed some differences in the obtained optimal hyperparameters of each prediction model compared to execution time modeling. We highlight in the following the main observed differences:

- MLP presents huge overfitting when increasing the size of the network in the case of execution time. However, small MLP networks are not enough to achieve good prediction of power consumption. Instead, they result in considerable underfitting. Consequently, we consider large MLP networks for power consumption modeling on the three edge GPUs.
- Linear kernels for SVR perform well for execution time modeling. However, they provide poor performances for power consumption modeling. Nevertheless, we have noticed that RBF kernels provide lower prediction error. Indeed, this can be explained by the observed non-linear correlation between input features and power consumption, for instance, between the number of FLOPs and power consumption (figure 6).

5.3.3 ***Rank-persevering:*** Figure 9 shows the calculated coefficients of correlation between measured and predicted values of power consumption. We can see that XGBoost, RF, and MLP provide the highest Kendall and

Pearson correlation coefficients in all the considered cases. These coefficients range from 0.7 to 0.9 for Kendall and from 0.8 to 0.9 for Pearson. On the other hand, SVR and Ridge Polynomial regression results are less correlated than the other results, especially for the Jetson Nano. The reason behind the low Kendall-tau values for Jetson Nano comes from the limited range of power consumption values. Generally, we have observed that obtained power consumption values are less distributed and tend to have very similar values than the execution time case where values have a higher degree of diversity. It is also worth noting that the overfitting problem of SVR appears clearly in the obtained correlation coefficient values for NCA. This observation is also proven by the obtained MAPE results reported in table 8, which indicate a considerable loss of generalization for NCA.

5.4 Memory Usage Prediction

Figure 15 gives the obtained MAPE values and their corresponding confidence interval of 95%. Table 10 summarizes the analysis of the five (05) prediction models in terms of MAPE, training, tuning costs, and prediction latency. Figure 11 reports the calculated rank correlation coefficients of all the obtained results for CNN memory usage estimations.

Table 10. Memory usage prediction models analysis: prediction error, training, tuning, and latency

Prediction Model	Test Data	MAPE			Training Time	Tuning Time	Prediction Latency		
		Nano	TX2	AGX			AGX	TX2	Nano
Poly	NIS	**9.73%**	**4.91%**	**4.38%**	**5.13 ms**	**6.09 mn**	**89.3 ns**	**140.5 ns**	**270.2 ns**
	NCV	**9.15%**	**5.01%**	**4.78%**					
	NCA	**6.43%**	**6.87%**	**5.47%**					
MLP	NIS	11.03	4.98%	5.85%	341.6 ms	13.8 hr	14.7 us	24.0 us	34.7 us
	NCV	14.83%	5.60%	4.92%					
	NCA	11.59%	9.04%	7.62%					
SVR	NIS	7.89%	6.65%	**3.62%**	101.2 ms	20.7 hr	96.6 us	121.3 us	249.6 us
	NCV	8.90%	**5.12%**	**3.64%**					
	NCA	7.82%	8.57%	6.78%					
RF	NIS	7.25%	6.22%	4.58%	1.62 s	3.1 hr	261.6 us	397.0 us	508.2 us
	NCV	8.15%	4.96%	6.83%					
	NCA	10.46%	9.18%	7.63%					
XGBoost	NIS	**9.03%**	**6.46%**	**4.10%**	**338.5 ms**	**7.2 mn**	**708.2 ns**	**933.8 ns**	**1.1 us**
	NCV	10.14%	**5.41%**	**5.72%**					
	NCA	7.60%	**8.73%**	**7.40%**					

5.4.1 ***Prediction error and generalization power:*** In contrast with the two previous performance metrics, where XGBoost and Ridge Polynomial regression provides similar results, Ridge polynomial regression outperforms XGBoost and the other models for memory usage. Indeed, training datasets for memory usage are different from those for execution time and power consumption. In fact, we have performed two reduction techniques on the training datasets for memory usage:

(1) Considering only the smallest and the largest input size for each CNN: From our experiments, we have noticed that the CNN memory usage does not vary significantly for two relative input sizes. Thus, we only consider the smallest and the largest image input sizes for each CNN, depending on the reported values in table 3.
(2) Selecting only the prediction CNN input features that impact memory usage.

These two data reduction techniques result in thoroughly different training datasets compared to the cases of execution time and power consumption.

From 15 and table 10, MLP, SVR, and RF provide similar performances in terms of MAPE values. As for the execution time and power consumption cases, reported prediction errors for the three edge GPUs are very close. This confirms the portability of our modeling methodology for new edge GPUs.

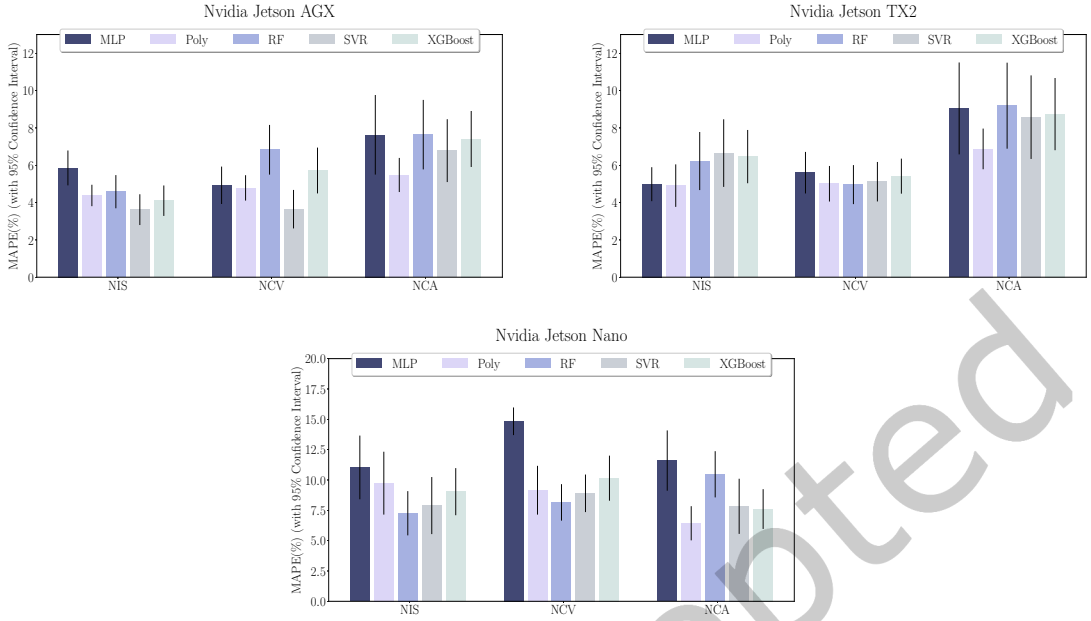

Fig. 15. Mean Absolute Percentage Error (MAPE) for memory usage on the three edge GPUs: Nvidia AGX, TX2, and Nano, respectively, with the corresponding 95% Confidence Interval. In this figure, we compare the five (05) prediction models for the three exploration spaces: New Image Sizes (NIS), New CNN Variants (NCV), and New CNN Architectures (NCA).

Table 11. Rank correlation coefficients between measured and predicted memory usage values for the three exploration spaces: NIS, NCV, and NCA, on different NVIDIA edge GPUs: Nano, TX2 and AGX.

Prediction model	Test data	Kendall coefficients			Pearson coefficients		
		Nano	TX2	AGX	Nano	TX2	AGX
Poly	NIS	**0.833**	**0.909**	**0.92**	**0.994**	**0.994**	**0.993**
	NCV	**0.871**	**0.909**	**0.92**	**0.995**	**0.994**	**0.993**
	NCA	**0.818**	**0.909**	**0.92**	**0.942**	**0.994**	**0.993**
MLP	NIS	0.858	0.912	0.905	0.994	0.992	0.988
	NCV	0.865	0.912	0.905	0.996	0.992	0.988
	NCA	0.784	0.912	0.905	0.912	0.992	0.988
SVR	NIS	0.824	0.875	0.923	0.991	0.987	0.993
	NCV	0.86	0.875	0.923	0.995	0.987	0.993
	NCA	0.79	0.875	0.923	0.908	0.987	0.993
RF	NIS	0.828	0.9	0.917	0.991	0.992	0.993
	NCV	0.848	0.9	0.917	0.991	0.992	0.993
	NCA	0.794	0.9	0.917	0.924	0.992	0.993
XGBoost	NIS	**0.82**	**0.885**	**0.907**	**0.991**	**0.991**	**0.993**
	NCV	**0.829**	**0.885**	**0.907**	**0.991**	**0.991**	**0.993**
	NCA	**0.783**	**0.885**	**0.907**	**0.926**	**0.991**	**0.993**

5.4.2 ***Training and tuning costs:*** Training and tuning costs for memory usage models are smaller than the execution time and power consumption models. These costs are correlated with the size and dimensionality of the training dataset. The larger the dataset is, the more time it takes to tune and train the prediction model. As we previously mentioned, the training dataset for memory usage is reduced, resulting in less effort and time to train the prediction models.

Nonetheless, the tuning and training costs follow the same tendency as execution time and power consumption. In contrast to power consumption modeling, smaller MLP networks perform better than larger networks. This is obvious given the dimensionality of the training dataset. Besides, for SVR, we have also noticed that RBF and Poly kernels deliver the best performances compared to the linear ones. Thus, SVR, MLP, and RF are the most challenging models to tune for the three performance metrics and the three edge GPUs.

5.4.3 ***Rank-persevering:*** From the reported values in figure 11, we can note that the rank is highly respected between the measured and predicted CNN memory usage values by the prediction models. Kendall and Pearson correlation coefficients range from 0.8 to 0.9 and from 0.95 to 0.98, respectively, which indicates a strong positive correlation between the two sets of data (i.e., measured and predicted memory usage). We must also note that the prediction models follow the same tendency for Kendall and Pearson correlation coefficients in all cases (i.e., exploration spaces and edge GPUs).

6 APPROACH ANALYSIS: ADVANTAGES AND LIMITATIONS

In this section, we discuss the main advantages and limitations of our work in terms of hardware and software adaptability.

6.1 Advantages of our approach:

Using a set of benchmarks that we built, we were able to evaluate and estimate CNN performances. Experimental results showed that:

- Our Machine Learning-based modeling methodology to characterize CNNs performances for edge GPUs gives predictions for latency, power consumption, and memory usage with good accuracy.
- Our resulting prediction models have been evaluated on a large set of CNN architectures and three (03) edge GPUs with different hardware resources.

Furthermore, Our benchmarks are composed of small and large, simple and complex, old and modern state-of-the-art CNNs and variants. In tables 12, 13, and 14, we present a comparison of our work with relevant prior studies on CNN performances modeling. We use the MAPE to assess the prediction error. For execution time, the prediction error of our models is the lowest for GPUs compared to related works. In addition, We have obtained the smallest MAPE for power consumption and memory usage compared to other related works.

Table 12. Execution time prediction: comparison between our proposed methodology and state-of-the-art works. We compare the prediction error according to the MAPE values

Ref.	CNNs	System	MAPE
[41]	NIN, VGG19M	TK1 CPU – Caffe	4.71%
		TK1 GPU – Caffe	23.70%
	NIN, VGG19M, SqueezeNet, MobileNet	TX1 CPU – Caffe	39.91%
		TX1 GPU – Caffe	31.51%
[10]	VGG16, AlexNet, NIN, Overfeat, CIFAR10-6conv	Titan X GPU – TensorFlow	7.96%
		GTX1070 GPU – TensorFlow	12.32%
	AlexNet, NIN	GTX1070 GPU – Caffe	16.17%
[64]	AlexNet, All-CNN-C, MobileNet, ResNet-18, SimpleNet, SqueezeNet, Tiny YOLO	RPi3 CPU – Caffe	5.02%
		RPi3 CPU – OpenCV	7.92%
		XU4 CPU – Caffe	3.25%
Our work	ResNets, MobileNets, EfficientNets, ShuffleNets, SENets, DenseNets, GoogleNet, Inception, SqueezeNets, MnasNets, DPN, Xception	Jetson Nano GPU – TensorFlow	**8.55%**
		Jetson TX2 GPU – TensorFlow	**9.33%**
		Jetson AGX GPU – TensorFlow	**7.86%**

Table 13. Power consumption prediction: comparison between our proposed methodology and state-of-the-art works. We compare the prediction error according to the MAPE values

Ref.	CNNs	System	MAPE
[41]	NIN, VGG19M, SqueezeNet, MobileNet	TX1 CPU – Caffe	39.08%
		TX1 GPU – Caffe	15.30%
[10]	VGG16, AlexNet, NIN, Overfeat, CIFAR10-6conv	Titan X GPU – TensorFlow	2.25%
		GTX1070 GPU – TensorFlow	8.40%
	AlexNet, NIN	GTX1070 GPU – Caffe	21.99%
[64]	AlexNet, All-CNN-C, MobileNet, ResNet-18, SimpleNet, SqueezeNet, Tiny YOLO	RPi3 CPU – Caffe	8.52%
		RPi3 CPU – OpenCV	7.24%
		XU4 CPU – Caffe	10.46%
Our work	ResNets, MobileNets, EfficientNets, ShuffleNets, SENets, DenseNets, GoogleNet, Inception, SqueezeNets, MnasNets, DPN, Xception	Jetson Nano GPU– TensorFlow	**4.23%**
		Jetson TX2 GPU – TensorFlow	**5.22%**
		Jetson AGX GPU – TensorFlow	**5.10%**

Table 14. Memory usage prediction: comparison between our proposed methodology and state-of-the-art works. We compare the prediction error according to the MAPE values

Ref.	CNNs	System	MAPE
[41]	NIN, VGG19M	TK1 CPU – Caffe	39.89%
		TK1 GPU – Caffe	34.33%
		TX1 CPU – Caffe	49.92%
		TX1 GPU – Caffe	40.94%
Our work	ResNets, MobileNets, EfficientNets, ShuffleNets, SENets, DenseNets, GoogleNet, Inception, SqueezeNets, MnasNets, DPN, Xception	Jetson Nano GPU – TensorFlow	**9.72%**
		Jetson TX2 GPU – TensorFlow	**5.93%**
		Jetson AGX GPU – TensorFlow	**4.91%**

6.2 Limitations of our approach

We have shown that our modeling methodology can be ported to any edge GPU to predict the performance of computer vision CNN. Furthermore, by using our proposed benchmarks and modeling methodology, the designer can quickly build prediction models by following our proposed modeling steps on new edge platforms, new use cases, or new metrics. Nevertheless, the ability to predict the performance of a set of CNNs on an entirely new edge GPU without training the performance prediction model on the new HW is not achievable because of the following reasons:

- In our work, we propose a modeling approach based on ML for predicting edge GPU's performances. However, as these embedded hardware platforms have been recently proposed in the market, the number of their hardware configurations is limited. The generalization for a new unknown HW without additional training or benchmarking is difficult.
- Modeling edge GPU platforms is complicated due to their complex integrated nature, where both the GPU and CPU share the same memory, and the differences in their micro-architectures.
- Building accurate analytical models for these edge GPUs requires modeling the holistic and hierarchical levels of the CNN execution stack. This stack comprises the CNN structure, the deep learning SDK, the compiler, and finally the hardware micro-architecture. As some of the details concerning the HW and SW are kept confidential by the GPU manufacturers (e.g. the NVIDIA CUDA compiler), this approach is very challenging.

We are also aware of the limitations of the proposed prediction models for non-computer vision CNN. The current prediction models are specific to computer vision applications as they were trained on CNN benchmarks for this purpose. However, we believe that the overall proposed modeling methodology, from data collection and feature extraction to ML-based modeling (figures 2 and 11) , can be applied to other CNN-based applications.

7 CONCLUSION AND FUTURE WORK

CNNs have achieved remarkable performances in vision-based applications at the cost of huge computational and memory complexities. Recently, tremendous effort has been made to design efficient hardware platforms to satisfy the computation and memory needs of CNNs. Edge GPUs are promising hardware platforms combining both acceleration capability and energy efficiency.

However, given the complexity and diversity of both CNN and edge GPU architectures, determining the appropriate matching between CNNs and edge platforms under real-time constraints is time-consuming and challenging. In this context, a priori prediction of CNN performances on edge GPU is a promising solution to determine the best CNN-edge hardware combination quickly.

In this work, we proposed a modeling methodology for CNN performance on edge GPUs. Our approach includes two main parts: First, we characterized the CNN architectures at a model-level granularity to extract the most impacting features. Second, we implemented and compared five of the most efficient Machine Learning algorithms to build our performance prediction models. The resulting prediction models can be used to predict execution time, memory usage, and power consumption of any new CNNs on the studied 3 edge GPUs without retraining the prediction models.

Our modeling methodology has been easily generalized over the three exploration spaces: NIS, NCV, and NCA, on the three different NVIDIA edge GPUs: Jetson Nano, TX2, and AGX. To evaluate our approach, we analyzed the performance of the prediction models using three metrics: 1) Prediction error and rank-preserving, 2) Prediction latency, 3) Training and tuning costs of the models and finally. We have demonstrated that XGBoost, Random Forest, and Ridge Polynomial regression estimate execution time with an average error of 10%. Ridge Polynomial regression and XGBoost give similar performances for power consumption with an average error of 6%. Finally, for memory usage, we have demonstrated that Ridge polynomial regression outperforms the rest of the prediction models with an average error of 8%. Experimental results demonstrated that MLP, SVR, and RF depicted the smallest overheads in tuning, training and prediction latency. On the opposite, both XGBoost and Ridge polynomial regression provided an acceptable trade-off between prediction performance and tuning/training time. They give also the lowest prediction latency, whereas RF has the highest prediction latency. Given that, the obtained prediction models can easily be integrated into a multi-objective optimization approach for design space exploration of CNN on edge GPUs.

We plan to extend our work by exploring different Deep Learning applications, hardware architectures, and high-performance deep learning SDK (e.g. TensorRT) to increase the flexibility and generalizability of our proposed approach. We also consider integrating the hardware as input into our prediction models for reconfigurable hardware platforms such as FPGA. Our prediction models can also be enhanced by characterizing the connections and dependencies between CNN layers, such skip and dense connections in ResNet and DenseNet or parallelism of inception blocks.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] Mohamed S. Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2020. Best of Both Worlds: AutoML Codesign of a CNN and its Hardware Accelerator. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218596

[3] Hervé Abdi. 2007. The Kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA* (2007), 508–510.

[4] Marcos Amarís, Raphael Y. de Camargo, Mohamed Dyab, Alfredo Goldman, and Denis Trystram. 2016. A comparison of GPU execution time prediction using machine learning and analytical modeling. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. 326–333. https://doi.org/10.1109/NCA.2016.7778637

[5] Yehia Arafa, Abdel-Hameed Badawy, Gopinath Chennupati, Atanu Barai, Nandakishore Santhi, and Stephan Eidenbenz. 2020. *Fast, Accurate, and Scalable Memory Modeling of GPGPUs Using Reuse Profiles.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3392717.3392761

[6] Mariette Awad and Rahul Khanna. 2015. Support vector regression. In *Efficient learning machines*. Springer, 67–80.

[7] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. *Pearson Correlation Coefficient.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1–4. https://doi.org/10.1007/978-3-642-00296-0_5

[8] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. arXiv:2101.09336 [cs.LG]

[9] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access* 6 (2018), 64270–64277. https://doi.org/10.1109/ACCESS.2018.2877890

[10] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. *NeuralPower*: Predict and Deploy Energy-Efficient Convolutional Neural Networks. In *Proceedings of the Ninth Asian Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 77)*, Min-Ling Zhang and Yung-Kyun Noh (Eds.). PMLR, Yonsei University, Seoul, Republic of Korea, 622–637. https://proceedings.mlr.press/v77/cai17a.html

[11] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 11285–11297. https://proceedings.neurips.cc/paper/2020/file/81f7acabd411274fcf65ce2070ed568a-Paper.pdf

[12] Jiasi Chen and Xukan Ran. 2019. Deep Learning With Edge Computing: A Review. *Proc. IEEE* 107, 8 (2019), 1655–1674. https://doi.org/10.1109/JPROC.2019.2921977

[13] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[14] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. 2017. Dual Path Networks. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/f7e0b956540676a129760a3eae309294-Paper.pdf

[15] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[16] Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. 2016. Mean Absolute Percentage Error for regression models. *Neurocomputing* 192 (2016), 38–48. https://doi.org/10.1016/j.neucom.2015.12.114 Advances in artificial neural networks, machine learning and computational intelligence.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[18] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (2020), 485–532. https://doi.org/10.1109/JPROC.2020.2976475

[19] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. 2018. Squeezenext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1638–1647.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, 630–645.

[22] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and their Applications* 13, 4 (1998), 18–28. https://doi.org/10.1109/5254.708428

[23] Arthur E. Hoerl and Robert W. Kennard. 1970. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* 12, 1 (1970), 55–67. https://doi.org/10.1080/00401706.1970.10488634

[24] Morteza Hosseini, Mohammad Ebrahimabadi, Arnab Neelim Mazumder, Houman Homayoun, and Tinoosh Mohsenin. 2021. A fast method to fine-tune neural networks for the least energy consumption on fpgas. *UMBC Student Collection* (2021).

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[26] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.

[27] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[28] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 0–0.
[29] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. 2020. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision* 12, 1–3 (2020), 1–308.
[30] Jongmin Jo, Sucheol Jeong, and Pilsung Kang. 2020. Benchmarking GPU-Accelerated Edge Devices. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 117–120. https://doi.org/10.1109/BigComp48618.2020.00-89
[31] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the Computational Cost of Deep Learning Models. In *2018 IEEE International Conference on Big Data (Big Data)*. 3873–3882. https://doi.org/10.1109/BigData.2018.8622396
[32] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. 2020. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review* 53, 8 (2020), 5455–5516.
[33] Mohsen Kiani and Amir Rajabzadeh. 2021. SDAM: a combined stack distance-analytical modeling approach to estimate memory performance in GPUs. *The Journal of Supercomputing* 77, 5 (2021), 5120–5147.
[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc.
[35] Yen-Lin Lee, Pei-Kuei Tsung, and Max Wu. 2018. Techology trend of edge AI. In *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 1–2. https://doi.org/10.1109/VLSI-DAT.2018.8373244
[36] Cheng Li, Abdul Dakkak, Jinjun Xiong, Wei Wei, Lingjie Xu, and Wen-mei Hwu. 2020. XSP: Across-Stack Profiling and Analysis of Machine Learning Models on GPUs. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 326–327. https://doi.org/10.1109/IPDPS47924.2020.00042
[37] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.
[38] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1529–1538.
[39] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. 2020. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565* (2020).
[40] Peiye Liu, Bo Wu, Huadong Ma, and Mingoo Seok. 2019. MemNet: memory-efficiency guided neural architecture search with augment-trim learning. *arXiv preprint arXiv:1907.09569* (2019).
[41] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. 2017. Modeling the Resource Requirements of Convolutional Neural Networks on Mobile Devices. In *Proceedings of the 25th ACM International Conference on Multimedia* (Mountain View, California, USA) *(MM '17)*. Association for Computing Machinery, New York, NY, USA, 1663–1671. https://doi.org/10.1145/3123266.3123389
[42] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-Sun Seo. 2020. Performance Modeling for CNN Inference Accelerators on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 4 (2020), 843–856. https://doi.org/10.1109/TCAD.2019.2897634
[43] Susmita Dey Manasi and Sachin S. Sapatnekar. 2021. DeepOpt: Optimized Scheduling of CNN Workloads for ASIC-Based Systolic Deep Learning Accelerators. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '21)*. Association for Computing Machinery, New York, NY, USA, 235–241. https://doi.org/10.1145/3394885.3431539
[44] Jiandong Mu, Wei Zhang, Hao Liang, and Sharad Sinha. 2020. Optimizing OpenCL-Based CNN Design on FPGA with Comprehensive Design Space Exploration and Collaborative Performance Modeling. 13, 3, Article 13 (jun 2020), 28 pages. https://doi.org/10.1145/3397514
[45] Fionn Murtagh. 1991. Multilayer perceptrons for classification and regression. *Neurocomputing* 2, 5 (1991), 183–197. https://doi.org/10.1016/0925-2312(91)90023-5
[46] Paulo Eduardo Nogueira, Rivalino Matias, and Elder Vicente. 2014. An Experimental Study on Execution Time Variation in Computer Experiments. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (Gyeongju, Republic of Korea) *(SAC '14)*. Association for Computing Machinery, New York, NY, USA, 1529–1534. https://doi.org/10.1145/2554850.2555022
[47] Nvidia. 2007. *Nvidia Profiler (Nvprof)*. Retrieved June 30, 2020 from https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview
[48] Nvidia. 2019. *Tegrastats utility*. Retrieved December 01, 2020 from https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra
[49] Eva Ostertagová. 2012. Modelling using Polynomial Regression. *Procedia Engineering* 48 (2012), 500–506. https://doi.org/10.1016/j.proeng.2012.09.545 Modelling of Mechanical and Mechatronics Systems.
[50] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. PALEO: A PERFORMANCE MODEL FOR DEEP NEURAL NETWORKS. In *ICLR*.
[51] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. 2020. Binary neural networks: A survey. *Pattern Recognition* 105 (2020), 107281. https://doi.org/10.1016/j.patcog.2020.107281
[52] Crefeda Faviola Rodrigues, Graham Riley, and Mikel Luján. 2017. Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 114–115. https://doi.org/10.1109/IISWC.2017.8167764

[53] Crefeda Faviola Rodrigues, Graham Riley, and Mikel Luján. 2017. Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*. 114–115. https://doi.org/10.1109/IISWC.2017.8167764

[54] Juan D. Rodriguez, Aritz Perez, and Jose A. Lozano. 2010. Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32, 3 (2010), 569–575. https://doi.org/10.1109/TPAMI.2009.187

[55] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. 2018. Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. 949–957. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.000-4

[56] Kevin Siu, Dylan Malone Stuart, Mostafa Mahmoud, and Andreas Moshovos. 2018. Memory Requirements for Convolutional Neural Network Hardware Accelerators. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. 111–121. https://doi.org/10.1109/IISWC.2018.8573527

[57] Dimitrios Stamoulis, Ermao Cai, Da-Cheng Juan, and Diana Marculescu. 2018. HyperPower: Power- and memory-constrained hyper-parameter optimization for neural networks. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 19–24. https://doi.org/10.23919/DATE.2018.8341973

[58] Qi Sun, Tinghuan Chen, Jin Miao, and Bei Yu. 2019. Power-Driven DNN Dataflow Optimization on FPGA. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–7. https://doi.org/10.1109/ICCAD45719.2019.8942085

[59] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[60] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.

[61] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

[62] Han Vanholder. 2016. *Efficient inference with tensorrt*. Retrieved June 30, 2020 from https://developer.nvidia.com/tensorrt

[63] Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. 2018. Optimum Selection of DNN Model and Framework for Edge Inference. *IEEE Access* 6 (2018), 51680–51692. https://doi.org/10.1109/ACCESS.2018.2869929

[64] Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. 2020. PreVIous: A Methodology for Prediction of Visual Inference Performance on IoT Devices. *IEEE Internet of Things Journal* 7, 10 (2020), 9227–9240. https://doi.org/10.1109/JIOT.2020.2981684

[65] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. 2019. Characterizing Deep Learning Training Workloads on Alibaba-PAI. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*. 189–202. https://doi.org/10.1109/IISWC47752.2019.9042047

[66] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. 2019. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701* (2019).

[67] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10734–10742.

[68] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. 2017. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 129–137.

[69] Huaizheng Zhang, Yizheng Huang, Yonggang Wen, Jianxiong Yin, and Kyle Guan. 2020. InferBench: Understanding Deep Learning Inference Serving with an Automatic Benchmarking System. *arXiv preprint arXiv:2011.02327* (2020).

[70] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2020. DNNExplorer: A Framework for Modeling and Exploring a Novel Paradigm of FPGA-Based DNN Accelerator *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 61, 9 pages. https://doi.org/10.1145/3400302.3415609

[71] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6848–6856.

[72] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, 8697–8710.

[73] Junhua Zou, Ting Rui, You Zhou, Chengsong Yang, and Sai Zhang. 2018. Convolutional neural network simplification via feature map pruning. *Computers & Electrical Engineering* 70 (2018), 950–958. https://doi.org/10.1016/j.compeleceng.2018.01.036

A APPENDIX: HYPERPARAMETERS TUNING OF THE PREDICTION MODELS

Designing accurate prediction models requires choosing optimal hyperparameters configurations for each ML algorithm before the training. To this end, we run an exhaustive grid search on the hyperparameters' search space. During this process, we evaluate the goodness of the explored hyperparameters via the K-fold cross-validation [54] to identify the optimal hyperparameter values and combinations. Afterward, the ML models with the optimal found hyperparameters are trained on their internal parameters using the training dataset. We give more details about the hyperparameters search space for each ML prediction algorithm in table 15.

Table 15. Hyperparameters of the prediction models and the default values before the tuning

Prediction model	Hyperparameters	Range	Default value
Poly	Degree	[2- 20]	2
	Lambda	[1e-8- 1.0]	1
SVR	Cost (C)	[0.5- 5000]	1
	Epsilon	[0.01- 5]	0.1
	Gamma	[0.01- 10]	scale
	Kernel	[linear- rbf- poly]	rbf
	Degree	[2- 20]	3
MLP	Hidden dim	[4- 72]	100
	Num. Layers	[1- 5]	1
	Activation	[identity- logistic- tanh- relu]	relu
	Optimizer	[sgd- adam]	adam
	Init. learning rate	[1e-4- 1e-1]	0.001
	Schedule. learning rate	[constant- adaptive]	constant
	Alpha	[1e-6- 1e-1]	0.0001
	Max. iterations	[100- 3000]	200
	Early Stopping	[False- True]	False
Random Forest	Num. estimators	[10- 500]	100
	Max. depth	[10- 300]	None
	Min. samples split.	[1- 10]	2
	Min. samples leaf	[1- 10]	1
	Max. features	[auto- sqrt- log2- None]	auto
	Bootstrap	[True- False]	True
XGBoost	Early Stopping	[True- False]	False
	Rounds	[100- 3000]	100
	Max. depth	[5- 20]	15
	Min. child weight	[5- 15]	5
	Sub.sample	[0.1- 1.0]	1
	Col. sample bytree	[0.1- 1.0]	1
	Gamma	[0.01- 10]	0.1
	Learning rate	[0.01- 0.9]	0.3