Performance Prediction for Convolutional Neural Networks in Edge Devices

Halima Bouzidi*, Hamza Ouarnoughi†, Smail Niar†, Abdessamad Ait El Cadi†
*École Nationale Supérieure d'Informatique, Algiers, Algeria
†Université Polytechnique Hauts-de-France, LAMIH/CNRS, Valenciennes, France
Email:fh_bouzidi@esi.dz
Email: firstname.lastname@uphf.fr

***Abstract*—Running Convolutional Neural Network (CNN) based applications on edge devices near the source of data can meet the latency and privacy challenges. However due to their reduced computing resources and their energy constraints, these edge devices can hardly satisfy CNN needs in processing and data storage. For these platforms, choosing the CNN with the best trade-off between accuracy and execution time while respecting Hardware constraints is crucial. In this paper, we present and compare five (5) of the widely used Machine Learning based methods for execution time prediction of CNNs on two (2) edge GPU platforms. For these 5 methods, we also explore the time needed for their training and tuning their corresponding hyperparameters. Finally, we compare times to run the prediction models on different platforms. The utilization of these methods will highly facilitate design space exploration by providing quickly the best CNN on a target edge GPU. Experimental results show that eXtreme Gradient Boosting (XGBoost) provides a less than 14.73% average prediction error even for unexplored and unseen CNN model architectures. Random Forest (RF) depicts comparable accuracy but needs more effort and time to be trained. The other 3 approaches (OLS, MLP and SVR) are less accurate for CNN performances estimation.**

***Index Terms*—CNN, GPU, Performance Modeling, Multiple Linear Regression, Multi-Layer Perceptrons, Support Vector Machine, Random Forest, eXtreme Gradient Boosting.**

I. Introduction and Motivations

Machine Learning (ML) approaches have recently become very popular for several use cases. Their fields of applications, in health, agriculture, transport, etc., are growing constantly. This increasing interest in ML in general and Convolution Neural Networks (CNN) in particular, could be explained by two main reasons. The first one is the availability of very powerful Hardware (HW) platforms in Internet of Things (IoT), edge, fog, Cloud computing and High performance computing (HPC) platforms. The second reason is the availability and the diversity of very large datasets on the Internet. These datasets allow to improve the training of ML algorithms and therefore enhance their accuracy.

Hence, the use of CNN has made it possible to obtain significant improvements in terms of accuracy and flexibility. However, their development and the choice of the best CNN for a given problem is difficult. Each month, a large number of complex and accurate CNN models are proposed. Consequently, finding the CNN which gives the best trade-off between accuracy and execution time is a tedious task [1].

For edge devices with limited computing power and real-time constraints such IoT or autonomous cars, choosing the best CNN implementation is problematic. In [2], authors have shown that CNN is not necessarily correlated with the number of FLoating-point OPerations (FLOPs) or the size of the CNN.

Having a design tool for rapid CNN performance estimation becomes crucial to reduce design time and to obtain a high-performance system. Early execution time estimation allows to quickly determine the best CNN implementation for a given application requirements and a given hardware (HW) constraints. Depending on the CNN to execute, the user may need different HW to minimize inference time. Such a tool can be used either to choose the best CNN for a given HW platform and/or to explore different HW platforms for a specific CNN.

In this paper, we focus on proposing a methodology to help the designer in choosing the most efficient CNN for a given HW platform. The CNN execution time estimator must provide a high accuracy, a low design time and a high level of flexibility to be adapted to different CNNs and HW platforms. In addition, the number of estimator hyperparameters must be reduced and easy to tune.

In this work, we compare some of the most successful state-of-the-art prediction algorithms for estimating the execution time of the CNN inference phase. As these algorithms have a set of (hyper)parameters, in the rest of the paper, they are called *Models*. The following ML-based models have been considered: Multiple Linear Regression using the Ordinary Least Squares (OLS), Multi-Layer Perceptrons (MLP), Support Vector Regression (SVR), Random Forest (RF), and eXtreme Gradient Boosting (XGBoost). Our work is not intended to propose new ML-based models but to compare the 5 models in order to analyze their strengths and weaknesses. For each of the studied models, we analyze prediction accuracy, time to tune and train the model and finally, time to run the prediction models on different Hardware platforms.

The remainder of this paper is organized as follows. Section II gives a literature review of CNN performance estimation and design space exploration. In section III, we first analyze and discuss the CNN features that have to be considered in the execution time prediction, then a survey of the used ML-based approaches is presented. Section IV is devoted to experimental results. Finally, conclusion and future work will be given in section V.

II. Related works

Motivated by speed and security purposes, there was recently a trend towards migrating ML applications from cloud and central computing to edge computing. To find a good matching between CNN requirements in terms of speed and energy consumption either in the training or in the inference phases, several projects have been conducted recently targeting edge platforms.

In [3], the authors compare: linear regression, support vector machines and random forests with a Bulk synchronous parallel (BSP) based analytical model. Machine learning approaches have been used to provide reasonable predictions without detailed knowledge of application code or hardware characteristics. The authors use profiling information from 9 benchmarks on 9 different GPUs. In opposite to our work, their method is not dedicated to ML applications. In addition, in our work we use a larger set of benchmarks and we obtain a better accuracy.

In [4], the authors propose a methodology to estimate training time for CNNs. In their approach, training time is seen as the product of the training time per epoch and the number of epochs which need to be performed to reach the desired level of accuracy. The deep learning network is decomposed in several parts and the execution time estimation operates on these parts. Timings for these individual parts are then combined to provide a prediction for the whole CNN execution time.

In [5], the authors propose an analytical framework to characterize deep learning training workloads in large AI clouds and clusters. Using different training architectures, the authors try to identify performance bottleneck for various DNN workloads. For the training phase, weight and gradient communication consumes almost 62% of the total execution time on average. Their simple analytical performance model is based on the key workload features and allows only to identify architectural bottlenecks.

PALEO [6] is another framework for estimating training execution time. In PALEO the number of floating point operations required for an epoch is multiplied by a scaling factor to obtain execution time for the training phase. However, PALEO does not take into account numerous other operations which do not scale linearly with the number of floating point operations and which has a big impact on execution time. In the literature we may also find several tools dedicated to power and memory optimization for edge HW platforms.

In [7], the authors proposed a multi-variable linear regression model to predict energy consumption of CNNs based on the number of SIMD instructions and main memory accesses. They used an Nvidia Jetson TX1 GPU and they obtained an average relative error of about 20%. Our profiling and analysis phases, detailed in Section III, have shown that these 2 parameters must be enhanced by others to obtain a more accurate estimation.

The idea behind tools in [8], [9] is to explore the hyperparameter space for NNs running on a given hardware platform and to estimate power and memory usages. In most of the existing projects, a learning-based polynomial regression approach is used. In our work, we do not treat power consumption estimation. Only tools for execution time estimation are presented and compared. However, as we will see in the following sections, the same approaches can be developed for power and energy consumption.

III. Proposed Approach

Figure 1 gives an overview of our proposed modeling methodology. The approach includes three main steps. The benchmarking step aims to define the benchmarks characteristics and determine the most impacting CNN features. To make the model simple, only these features will be taken into account in the models. In the data collection step, we extract the data used in the elaboration of the CNN execution time prediction. Finally, the modeling step details the used ML-models, their training process with hyperparameters tuning and their evaluation. As explained in the following section, each of the considered models includes a set of *hyperparameters* and a set of *parameters*. The hyperparameters are different from one model to another. They draw the model architecture. The parameters produce one instance of the obtained model and explain the manner by which the model is implemented for the prediction. As example for MLP, the hyperparameters correspond to the number of neurons, the number of hidden layers, the activation function, etc. The parameters for MLP correspond to the weights and the biases values used in the model. More details will be given in sub-Section III-C.

A. Problem Formulation

Predicting CNNs inference time on GPU platforms can be formulated as follows: As inputs, we have a CNN (noted CNN_i) characterized by a set of n features $f_1, f_2, \ldots, f_n$, and a GPU platform (noted GPU_j) represented by a set of m Hardware characteristics $c_1, c_2, \ldots, c_m$. The number of convolutional and fully-connected layers, the image size, and the number of neurons are examples of CNN features. Platform parameters may correspond to the number of cores in the GPU, the memory size, the clock frequency, etc.

An inference time prediction function (noted T) is a mapping function from CNN_i and GPU_j to $\mathbb{R}_+$

$$T : CNN_i, GPU_j \rightarrow \hat{y}$$

$$\hat{y} = T(f_1, f_2, \ldots, f_n, c_1, c_2, \ldots, c_m) \tag{1}$$

where $\hat{y}$ is the predicted inference time of CNN_i on the GPU_j.

In this paper, due to the lack of space, we only present the case of two different edge GPUs, namely Nvidia Jetson AGX Xavier [10] and Nvidia Jetson TX2 [11].

The same modeling methodology has been used on each edge GPU platform and two sets of models have been obtained. The equation 1 becomes:

$$\hat{y} = T_{GPU_j}(f_1, f_2, \ldots, f_n) \tag{2}$$

where $GPU_j \in \{AGX, TX2\}$.

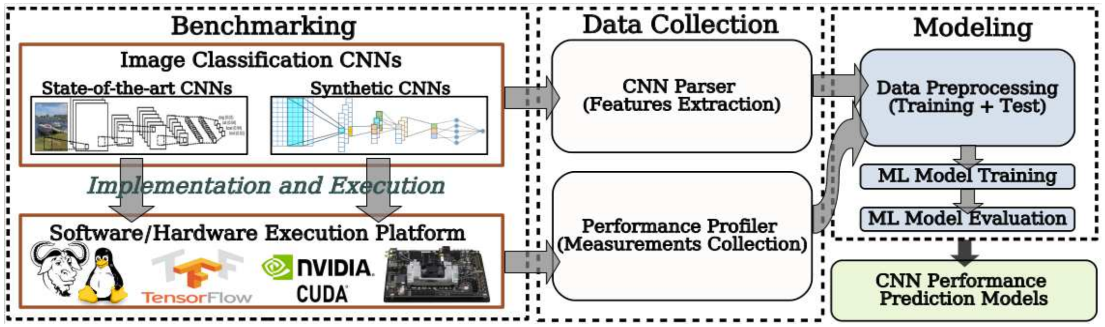

Fig. 1: Modeling methodology for CNNs execution time prediction models.

In this context, our approach aims to answer the following questions:

1) What are the most important CNN features that impact inference time? Our answer is given in sub-section III-B.
2) What are the most relevant modeling algorithms that give the best accurate predictions of CNNs inference time? Our answer is given in sub-section III-C.

Our modeling approach is based on finding the relationship between the input CNN features and observed inference time. For this purpose, we rely on regression models which are part of supervised ML algorithms.

B. CNN Features for our Prediction Models

CNNs inference time is mainly impacted by the following factors:

1) Computational complexity, which corresponds to GPU cores activities;
2) Memory requirements, which corresponds to read and write memory operations;
3) CNN internal properties, which corresponds to dependencies between computation operations and memory operations.

These features are detailed below.

1) Computational complexity: The total number of FLOPs is used to measure the computational complexity when performing CNNs inference. As reported in table I, this number is highly dominated by the number of operations performed in convolutional layers. Convolutional layers represent the bottlenecks of computations in CNNs. Our experimentation shown in figure 2 confirms that CNNs inference time is not linearly correlated to the number of FLOPs. We can then conclude that considering only the computational complexity is not enough to predict accurately the CNN inference time.

2) Memory requirements: Memory activities have a significant impact on the execution time on GPUs. However, it's hard to extract the information about memory activities and requirements without profiling the CNNs on the GPU platform. This solution must be avoided during the prediction as it's a time consuming task and will also add a significant overhead to the prediction latency. To overcome this problem, we rely on some specific characteristics of CNNs that are correlated to memory activities. Our experiments show that the major memory requirements of CNNs can be devoted mainly to 3 factors:

1) Reading CNNs parameters, i.e weights and biases,
2) Reading the input data, writing the output results and
3) Reading and writing the intermediate data of the hidden layers, i.e activations.

During the CNN inference, convolutional filters and activations are constantly accessed which increases the inference time [12]. Moreover, this time increases when activations and weights can not be mapped entirely in cache memories. Cache misses considerably increase the CNN inference time. Given the above facts, we assume that both weights and activations are highly impacting memory activities when performing CNNs inference.

3) CNN internal properties: In addition to computational complexity and memory requirements, other properties related to CNNs internal architecture impacting inference time have to be taken into account. We have also considered the **Weighted sum of neurons** in convolutional layers. This metric is calculated by multiplying the number of neurons in convolutional layers by the filter size: $height \times width \times depth$. The idea here is to give more importance to neurons with large filter sizes. The number of neurons in fully connected layers is taken as it is, because the neuron is associated to a single weight.

To select the most important CNN's features in the execution time prediction, we used the F-score metric of the XGBoost algorithm. This technique is used for all of the prediction models detailed in section III-C. The considered features and their corresponding impacts are depicted in figure 3.

C. Modeling Approaches

Five ML-based algorithms have been used to design the CNN inference time prediction models:

1) Multiple Linear Regression using the Ordinary Least Squares (OLS)
2) Multi-Layer Perceptrons (MLP)
3) Support Vector Regression (SVR)

TABLE I: FLOPS ESTIMATED BY TENSORFLOW PROFILER OF SOME STATE OF THE ART CNNS. (**B:** BILLIONS, AND **M:** MILLIONS).

CNN Model	Conv2D	Add	Mul	Pooling
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

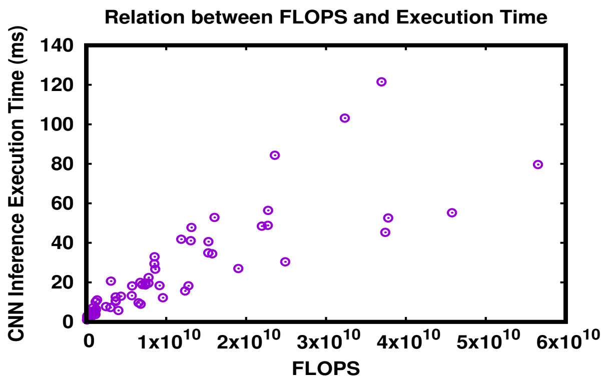

Fig. 2: Execution time versus floating-point operations (FLOPs). This figure demonstrates that using only the numbers of FLOPS is not sufficient to estimate execution time. This figure gives the execution time of some of our CNN benchmarks detailed in Section IV ranked by their FLOPs.

TABLE II: EVOLUTION OF THE ADJUSTED R^2 DURING THE STEPWISE LINEAR REGRESSION. FEATURES ARE ADDED TO THE OLS PREDICTION MODEL ONE BY ONE, FROM LEFT (MOST IMPORTANT FEATURE) TO RIGHT (LESS IMPORTANT FEATURE) OF THE TABLE

Features	Total number of FLOPs	Sum of activations	Weighted sum of neurons	Number of convolution parameters	Total number of layers	Input image size	Number of fully connected parameters	Number of batch normalization parameters	Number of batch normalization layers	Number of convolutional layers	Number of fully connected layers
Adjusted R^2	0.970	0.985	0.987	0.988	0.988	0.988	0.988	0.988	0.988	0.988	0.988

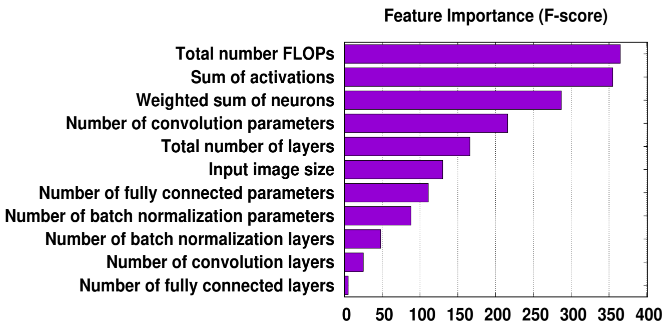

Fig. 3: The feature importance calculated by XGBoost. Features with a high F-score are the most impacting on the CNNs inference time.

4) Random Forest (RF)
5) eXtreme Gradient Boosting (XGBoost)

In order to tune each model's hyperparameters, we applied an exhaustive grid search. During this process, we perform a K-fold cross validation [13] to select the best hyperparameters values and combination. Once the hyperparameters are tuned, the training of the chosen ML algorithms is performed on the parameters using the training dataset. Finally, the obtained prediction models are evaluated using the test dataset. We give an overview of the ML algorithms used in our approach in the following.

1) Multiple Linear Regression using the Ordinary Least Squares (OLS): Multiple linear regression approaches use a linear function to model the relationship between a target variable (y) and n input features ($X_1, X_2, \ldots, X_n$):

$$y = \alpha_n X_n + \alpha_{n-1} X_{n-1} ... + \alpha_2 X_2 + \alpha_1 X_1 + \beta + \epsilon \quad (3)$$

In this equation $\alpha 1, \alpha 2, \ldots, \alpha_n$ and β are the parameters of the MLR. They are fixed during the training phase in such a way to minimize the difference (ϵ) between estimated ($\hat{y}$) and real (y) values of the inference execution times.

Among the variety of Multiple Linear Regression algorithms, we have chosen the most frequently used Ordinary Least Squares regression (OLS) [14]. In our study, we have applied a stepwise regression to choose the best combination of features with the highest impact on CNNs inference time. The results of the stepwise regression are given in table II. The features are added based on their importance showed in the figure 3.

We notice that the model has achieved a highest adjusted R^2 (98.8%) with only four input features, namely : FLOPs, sum of activations, weighted sum of neurons, and number of convolutional layers parameters. The reasons behind choosing Multiple Linear Regression for CNNs inference time prediction are: 1) their simplicity in the implementation, and 2) their short training time. However, this approach is not accurate when the linear relationship between input features and the target variable, execution time in our case, is not valid. To overcome this limitation, other algorithms which have the ability to model both linear and non-linear relationships have been implemented and compared. These algorithms are presented in the following sub-sections.

2) Multi-Layer Perceptrons: The Multi-Layer Perceptrons (MLP) [15] is a part of Artificial Neural Networks (ANN). 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. Choosing the good MLP architecture is done by fixing the hyperparameters, such as the number of layers, number of neurons, types of activation function, etc. We rely on the grid search method with the K-fold cross validation to choose the best hyperparameters values. Once these hyperparameters are fixed, parameters values of w_i and b are fixed at the training phase.

3) Support Vector Regression: Support Vector Machine [16] is one of the most powerful data driven algorithms used for classification and regression problems. The SVR deals with non-linearity by using kernels which are functions that map the input data from the original space to a higher dimensional space where features can be linearly separable. The most important hyperparameters for the SVR are:

- Kernel type: linear, polynomial, sigmoid, and radial basis function (RBF).
- Gamma: the kernel coefficient for the polynomial, sigmoid and RBF kernels.
- Cost (C): regularization factor which controls the trade-off between the training error and the generalization of the model.
- Epsilon, which defines the interval of errors within which no penalty is applied to training loss function.

These hyperparameters have been tuned using the grid search method.

4) Random Forest: Unlike the above methods based on the best line or hyperplane fitting, Random Forest (RF) uses decision trees. RF 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. We tune the following hyperparameters:

- Number of predictors needed to train the model,
- Maximum depth of each decision tree,
- Minimum number of samples required to split nodes,
- Minimum number of samples required for a leaf node,
- Maximum number of features to consider when splitting the nodes.

We have also used the bootstrap method for data sampling.

5) eXtreme Gradient Boosting: eXtreme Gradient Boosting (XGBoost) [17] is a decision-tree based algorithm. Unlike Random Forest, XGBoost is based on the boosting technique. The boosting technique makes predictions from weak predictors that are arranged sequentially: the first predictor is trained on the entire dataset, where the subsequent predictors are trained on the residuals of the prior predictors. This technique helps to focus on the mispredicted values. The algorithm uses the gradient descent algorithm to minimize prediction errors. As explained in Section III-B, we use first the feature importance calculated by XGBoost to select the most impacting features (see figure 3). These features are then used by the performance estimator. XGBoost has several hyperparameters to tune, which can be categorized into three groups:

- General hyperparameters,
- Booster hyperparameters,
- Learning hyperparameters.

Once the optimal combination of hyperparameters is obtained, we perform a full training on the entire training dataset, with the early stopping technique.

IV. Experimental Results

This section details the followed evaluation methodology and the obtained evaluation results. We used two Nvidia GPU platforms, namely the Jetson TX2 and the Jetson AGX Xavier. The hardware specifications of each platform is described in the table III. We have configured the platforms to profiling mode (maximum power mode) in order to minimize the host CPU interference. We have used the same underlying software configuration in the two edge GPU platforms. CNNs have been implemented on GPU using the Keras 2.3.1 API with TensorFlow 1.14 as backend [18]. This framework is running on top of Cuda version of 10.0 and a cuDNN version of 7.5.3. The host operating system in both platforms is Linux Ubuntu 18.04.3 LTS with a kernel 4.9.149-tegra.

TABLE III: Hardware platforms used in the experiments

Hardware feature	**Jetson TX2**	**Jetson AGX Xavier**
CPU(ARM)	6-core Denver and A57 2 GHz	8-core Carmel 2.26 GHz
Memory	8 GB 128-bit LPDDR4	16 GB 256-bit LPDDR4x
Memory bandwidth	58.4 GB/s	137 GB/s
GPU	256-core Pascal 1.3 GHz	512-core Volta 1.37 GHz + 64 tensor cores
Power	7.5W/15W	10W/15W/30W

A. Evaluation Methodology

As shown in figure 1, our modeling process is subdivided into three main steps: 1) Benchmarking, 2) Data collection and 3) Modeling.

1) Benchmarking step: The benchmarking step is mainly based on profiling CNNs inference on different GPUs. The experiments have been designed based on executing state-of-the-art image classification CNNs by varying the parameters presented in table IV.

As shown in table IV, we have implemented the state-of-the-art CNNs dedicated to image classification. CNN architectures

TABLE IV: Details of the benchmarks used in the experiments. In total, we have 2056 and 1975 (respectively) input-data for AGX and TX2 (respectively) for 5 performance estimation models: 70% have been used for training and 30% for tests and accuracy calculations

CNN Architectures	# CNN variants	Input Image Sizes (squared)
GoogleNet	1	[224,240,256,299,320,331,448,480, 512,568,600,720,800,896,1024]
Inception V3	1	[75,90,112,128,150,224,240,256, 299,320,331,448,480,512,568,600, 720,800,896,1024]
InceptionResNet V2	1	
DPN	4	[32,56,64,75,90,112,128,150,224, 240,256,299,320,331,448,480,512, 568,600,720,800,896,1024]
DenseNet	5	
Xception	1	
EfficientNet	4	[32,56,64,75,90,112,128,150,224, 240,256,299,320,331,448,480,512, 568,600,720,800,896,1024,1200,1600]
MNASNet	5	
ResNet V1	12	[32,56,64,75,90,112,128,150,224, 240,256,299,320,331,448,480,512, 568,600,720,800,896,1024, 1200,1600,1792,2400]
ResNet V2	7	
MobileNet V1	4	
MobileNet V2	5	
MobileNet V3	4	
ResNext	2	
SENet	6	
ShuffleNet V1	4	
ShuffleNet V2	3	

such as: Inception, ResNet, MobileNet, etc., have been tested on the two GPU platforms. An FP-32 bits representation for the weights and tensors has been used. The weights of these CNNs have been set randomly as their values have no impact on the inference time. We varied three factors:

1) Input Image Sizes: Here the impact of the image size on the inference time is studied. We tested the frequently used image sizes in the literature from 32*32 to 2400*2400 pixels depending on the CNN.
2) CNN Variants: Different variants of the same CNN architecture are considered in order to extend our dataset. Some of the considered CNN, such as ResNet V1, have up to 12 different variants.
3) CNN Architectures: Finally, we consider different architectures of CNN to quantify their impact on the inference time.

In total, we obtained a dataset of 2056 and 1975 (respectively) on AGX and TX2 (respectively) as inputs for our performance estimation models. This difference is due to the fact that the TX2 GPU platform has a smaller memory and some of the benchmarks couldn't be executed. In the experiments, 70% of the input-data has been used for training and 30% for tests and accuracy measurements. In the experimental results section, we will evaluate the accuracy of each of the 5 models, when exploration is realized in 3 groups.

2) Data collection step: The dataset used for CNNs inference time modeling is collected from two sources:

- CNNs implementation descriptions: we have developed a parser module that takes CNNs implementation as input and gives their important feature values (see figure 3).
- Performance measurements: CNNs inference times measurements have been collected using the Nvidia profiling tool Nvprof [19]. Each inference has been ran 100 times on 100 randomly chosen images in order to minimize the profiling overhead.

At this point the dataset is collected and ready to use in the modeling step.

3) Modeling step: Figure 4 details the modeling steps followed to obtain the 5 prediction models.

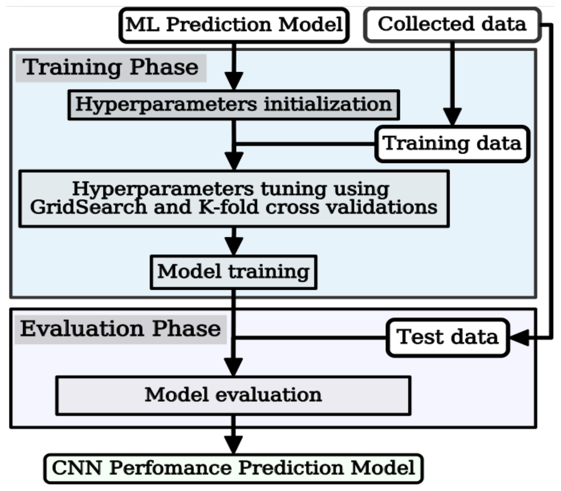
ML Prediction Model
Collected data
Training Phase
Hyperparameters initialization
Training data
Hyperparameters tuning using GridSearch and K-fold cross validations
Model training
Evaluation Phase
Test data
Model evaluation
CNN Perfomance Prediction Model

Fig. 4: The process of constructing the prediction models.

As shown in the figure 4, our modeling process has two inputs:

1) **ML prediction model name**, which corresponds to one of the five ML algorithms described in section III-C.
2) **Collected data**, which is the dataset obtained in the data collection step as described in Table IV.

The collected data is split into training data and test data. During the training phase, the search space for the hyperparameters is firstly initialized for each prediction model before being tuned using the training data. The tuning is realized using grid search and K-fold cross validation techniques in order to select the best values of hyperparameters. Finally, the prediction models are trained with the obtained optimal hyperparameters.

To evaluate the obtained models on different configurations we have also split the test data into three exploration spaces:

- **Performance estimation of New Image Sizes (NIS):** In this first group of experiments, we evaluate our 5 models on state-of-the-art CNN models with new input sizes. We varied the input size from 32*32 to 2048*2048 pixels. We obtained different values of FLOPs, sum of intermediate activations, sum of neurons and eventually different number of parameters. The number and the type of layers remain the same.
- **Performance estimation of New CNN Variants (NCV):** In this second group of experiments, we measure performance estimation when new variants of the same CNN is

considered. Based on an original CNN architecture, new CNN variants have been obtained by changing the features such as the number of layers and the used operators. If we consider ResNet V1 as example, we trained our predictors on ResNetV1-20 to ResnetV1-100 and we predict inference time for ResNetV1-200. This leads to new numbers of: FLOPs, activations, neurons, parameters and layers.

- **Performance estimation of New CNN Architectures (NCA):** In this group of experiments, we evaluate our predictors on new hand crafted synthetic CNN models. We randomly generate synthetic CNN architectures where the input size, number of layers, type of layers, number of filters, type and parameters of convolutions, parameters of fully-connected layers, and type of pooling and batch normalization layers, have been randomly set. This exploration space is the most difficult as we evaluate the capacity of the 5 models to predict the performances of completely new CNN architectures not included in the training dataset. The aim of this 3^{rd} group of experiments is to measure the capacity of the 5 models to estimate execution time of completely new and unseen CNN architectures only by characterizing them through their features.

Inference execution time accuracy is measured using the Mean Absolute Percentage Error (MAPE).

B. Results and Discussion

After detailing the evaluation methodology, this section presents and discusses the obtained modeling results in three perspectives: First, we discuss the predicted CNN inference times using the 5 ML prediction models compared to the measured ones. Second, the Mean Absolute Percentage Error (MAPE) is presented to evaluate the accuracy of each ML model. We also discuss the hyperparameters configuration. In the third part of this section, we compare execution times for training and for tuning, the number of tested hyperparameters' configurations and finally the prediction latency which corresponds to the execution time of the 5 prediction approaches on 3 different Hardware platforms: on the Jeston AGX GPU, on the Jetson TX2 GPU and on a development station Intel Xeon. Implementing our predictors on edge GPU platforms will allow to adapt the executed CNN to application constraints at run time.

1) Predicted vs Measured CNN inference time: Figures 6, 7, 8, 9, and 10 present the comparison between the predicted (y axis) and the measured (x axis) CNN inference time on the AGX platform. The comparison has been realized for NIS, NCV (on the left of the figures) and NCA (on the right of the figures). The used dataset has been detailed in section IV-A and Table IV.

For NIS and NCV, the predicted execution times are very close to the measured values, in all of the prediction models except for OLS (see figure 6). As we have trained our prediction models on different CNN variants and different input image sizes, the prediction models have been able to capture the correlation between different CNN variants and input sizes. We can also notice that OLS overestimates the CNN inference time especially for high execution time values. This is due to the non-linearity between input features and the CNN inference time where the computation complexity and memory requirements are very high. This result confirms also that the inference times could not be accurately estimated by using a linear regression .

For NCA, the difference between the predicted and the measured execution times are higher than for NIS and NCV, in particular for MLP, OLS and SVR prediction models. As explained in the previous section, the reason behind this drawback is the fact that in NCA, new and randomly generated CNNs are explored. For these reasons, the predictions models are less accurate for these unexplored and unseen CNN architectures.

In the appendix of the paper, figures 11, 12, 13, 14, and 15 show the results for the NVIDIA Jetson TX2 platform. We observe the same conclusions as for the NVIDIA AGX.

2) Prediction accuracy and hyperparameters configuration analysis : To evaluate the accuracy of our prediction models, the Mean Absolute Percentage Error (MAPE) has been chosen. Figure 5 and Table V give the obtained MAPE for the 5 studied prediction models for NIS, NCV and NCA. The prediction models use data obtained by profiling the benchmarks (Table IV) on two edge GPU platforms (see section IV-A1).

From figure 5 and Table V, we can note that the MAPE average values are mostly between $\sim 7\%$ in the best case and $\sim 26\%$ in the worst case. We can also notice that the lowest MAPE values have been obtained for NIS and NCV.

In general we can say that for NIS, NCV and NCA, XGBoost outperforms the other prediction models and offers the lowest MAPE values. XGBoost is among the most powerful ML algorithms. The Boosting technique, to enhance the prediction accuracy through many estimators arranged sequentially, makes it very accurate. However, we noticed that XGBoost is sensitive to hyperparameters values. These values need to be set carefully in order to achieve the best performances.

RF is composed of different decision trees trained on random subsets of training data samples and features. This property helps to reduce the variance error. Nevertheless, prediction approaches based on ensemble learning, such as RF, need to be trained on different scales of data in order to obtain an accurate mapping of features and execution times. In addition, by increasing the number or the depth of the decision trees, the prediction accuracy converges according to our the experiments. These 2 factors increase the complexity of the training and the latency of the prediction in RF. For this reason, when a short interval of time for tuning and training is desired, XGBoost will be more efficient than RF.

MLP has generally good performances with a slight loss of generalization for NIS and NCA. This can be due to its nature that tends to overfit data. Furthermore, MLP is very sensitive to the variation of hyperparameters, which makes it very hard to tune. The size of the MLP network has an important impact

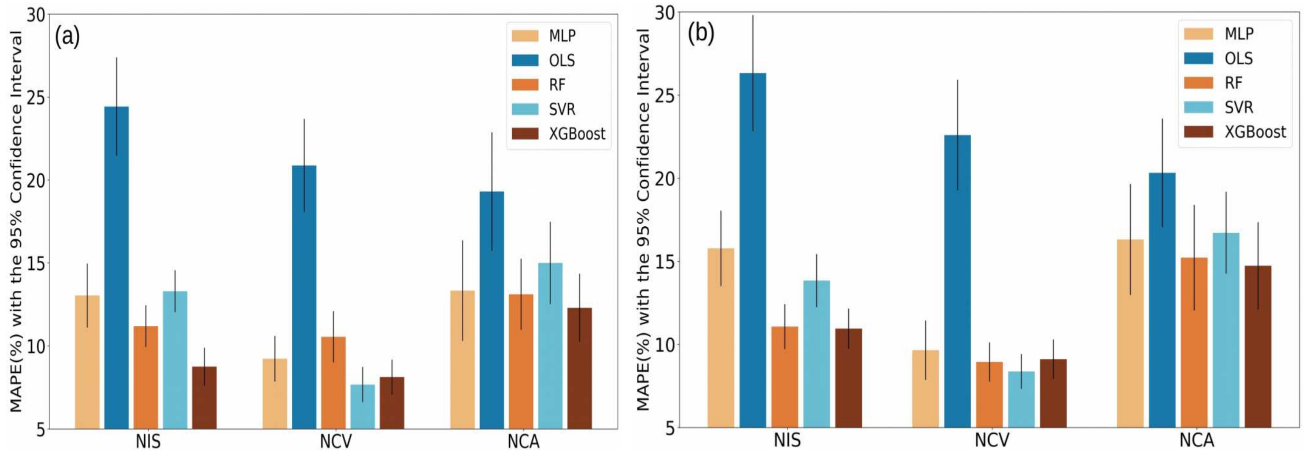

Fig. 5: Mean Absolute Percentage Error (MAPE) for the Nvidia AGX **(a)** and TX2 **(b)** GPU platforms with the corresponding 95% Confidence Interval. In this figure we compare the 5 prediction models for exploring: New Image Sizes (NIS), New CNN Variants (NCV) and New CNN Architectures (NCA).

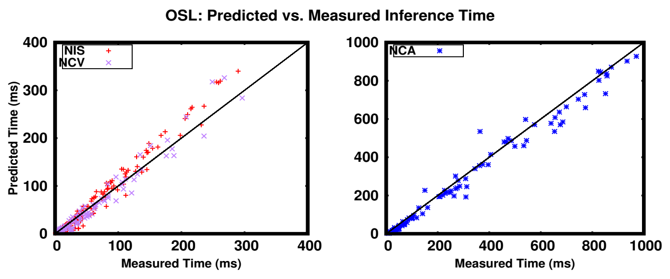

Fig. 6: Predicted vs. Measured Inference Time for OLS (AGX).

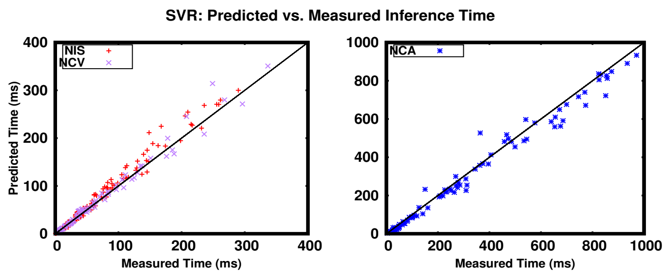

Fig. 7: Predicted vs. Measured Inference Time for SVR (AGX).

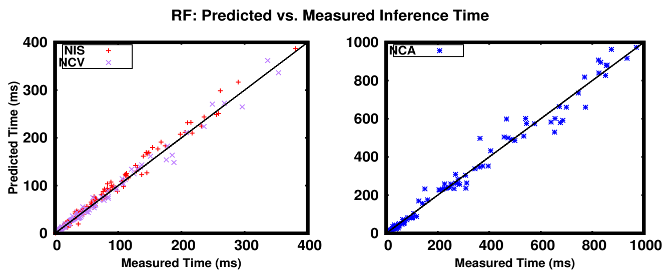

Fig. 8: Predicted vs. Measured Inference Time for RF (AGX).

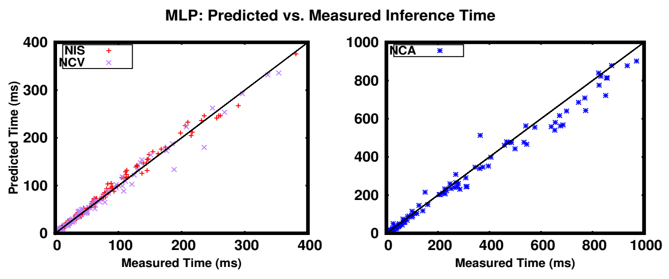

Fig. 9: Predicted vs. Measured Inference Time for MLP (AGX).

TABLE V: PREDICTION MODELS ANALYSIS: ACCURACY, TRAINING, TUNING, AND LATENCY

		MAPE		Training time		Grid Search execution time		Total number of hyperparameters		Prediction latency		
Prediction Model	Test Data	AGX	TX2	AGX	TX2	AGX	TX2	AGX	TX2	AGX	TX2	Intel Xeon
OLS	NIS	24.43%	26.32%	**8.14 us**	**7.3 us**	0		0		319.6 ns	468.42 ns	113.93 ns
	NCV	20.87%	22.95%									
	NCA	19.30%	20.32%									
MLP	NIS	13.04%	15.78%	1.53 s	722 ms	10.35 hr	11.82 hr	14400	15840	22.65 us	30.90 us	9.95 us
	NCV	9.23%	9.65%									
	NCA	13.34%	16.31%									
SVR	NIS	13.30%	13.84%	127 ms	191 ms	23.61 hr	21.43 hr	18144	18144	41.79 us	52.14 us	20.76 us
	NCV	**7.67%**	**8.37%**									
	NCA	15.00%	16.72%									
RF	NIS	11.19%	11.07%	4.93 s	2.01 s	4.69 hr	4.22 hr	71280	71280	1.03 ms	1.45 ms	393.3 us
	NCV	10.55%	**8.95%**									
	NCA	13.11%	15.22%									
XGBoost	NIS	**8.75%**	**10.95%**	163 ms	914 ms	**11.27 mn**	**15.17 mn**	**237**	**238**	2.03 us	3.49 us	175.20 ns
	NCV	**8.12%**	**9.11%**									
	NCA	**12.29%**	**14.73%**									

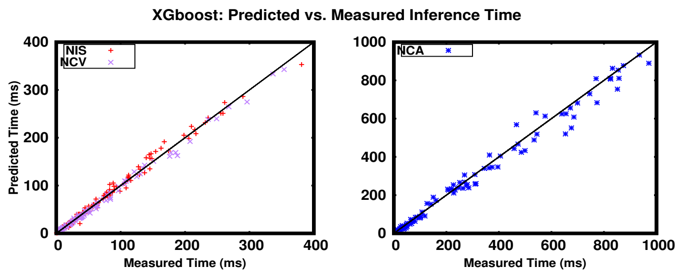

Fig. 10: Predicted vs. Measured Inference Time for XGBoost (AGX).

on the accuracy. We noticed that large MLP networks are prone to overfitting compared to the small ones.

SVR is less accurate for NIS and NCA compared to the aforementioned models. The variation in the MAPE values in the three exploration spaces is quite high which can be interpreted as overfitting. In terms of hyperparameters configuration, SVR is sensitive to the type of kernels and the cost (C). According to our results, linear kernels perform the best, whereas the very small values of C, lower than 1, lead to a huge loss of generalization.

As expected, OLS has the highest MAPE values because of its limited capacity to capture the different patterns in the training data. Moreover, the inclusion of non relevant features in OLS can drastically decrease the model's accuracy. In this case, techniques for regularization such as LASSO and Ridge are recommended to improve the model performances. Due of time limitation, this point is considered as a possible extension.

If we compare the results of the two GPU platforms, Nvidia AGX and TX2, we notice that the MAPE is very similar. This means that our modeling approach is easily adaptable to other platforms in order to obtain execution time prediction.

3) Comparison of the 5 models in terms of training time, tuning cost and latency: Table V summarizes the obtained statistics of the 5 tested modeling approaches.

In addition to MAPE average values, table V gives the training and tuning time of each prediction model for AGX and TX2 in Google Colaboratory [20]. We can observe, that except for SVR and XGBoost, the prediction models require more time to be trained for AGX than for TX2. This is due to the fact that the higher resource capacity of AGX allows to run more CNNs which gives a largest training data compared to TX2. Table V shows also that tuning the model hyperparameters is time consuming task, especially for models with high number of hyperparameters.

Tested hyperparameters values in grid search for the AGX and for the TX2 are different. This is due to the fact that CNN execution times ranges are different. For the MLP and XGBoost, as both of them use the gradient descent algorithm, the training convergence for AGX and for TX2 is different.

To quantify the execution time of the 5 models, they have been executed on an AGX, TX2 and Intel Xeon based Processor development station. Results are given in the three rightmost columns in Table V. The comparison of the execution latency is important to measure the performance of the 5 models to explore a large number of CNN architectures in a limited interval of time. In addition, when an online tuning of the CNN or the Hardware platform is needed, having a short execution time of the prediction model is interesting. We note that in terms of speed, OLS outperforms all of the other models. This is explained by the simplicity of OLS which is a simple linear equation. However OLS is the less accurate. XGBoost has a smaller execution time compared to MLP, SVR and RF. Which makes this technique very interesting due to the good trade-off in terms of accuracy and latency. The highest latency of RF is due to the complexity of exploring the decision trees included in the Random Forest.

V. Conclusion

Edge computing is one of the area targeted by CNN-based applications. Both CNNs and edge computing are highly growing and frequently changing domains. Finding the best matching between CNN model architecture and Hardware of the edge device under real time constraints is a very time consuming and complex task. In this paper, we compared five state of the art ML-based models for CNN inference time prediction on edge GPUs. These 5 models are useful for an early performance estimation of CNN-based applications. Using these models, rapid design space exploration will be possible to guide the designer to an efficient CNN model and/or to the most adequate Hardware platform.

We demonstrated that XGBoost and RF gave good execution time predictions, with an average accuracy close to 92% for NIS and NCV and 86% for NCA. In terms of trade-off between: accuracy, model tuning complexity and prediction latency, XGBoost is the best approach. Exploring new synthetic CNNs, i.e called NCA in our paper, is more complex and less accurate than the other two spaces: NIS and NCV.

As future work, we plan to extend our comparison to explore additional CNNs' characteristics that may improve our prediction models. We also plan to extend our ML-based models to predict not only CNNs inference execution time but also energy consumption and resource utilization. Finally, we will also integrate the hardware configuration as input in our models to perform cross platform predictions.

References

[1] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[2] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018.

[3] M. Amarís, R. Y. de Camargo, M. Dyab, A. Goldman, and D. Trystram, "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)*, 2016, pp. 326–333.

[4] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 3873–3882.

[5] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia, "Characterizing deep learning training workloads on alibaba-pai," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 189–202.

[6] H. Qi, E. R. Sparks, and A. Talwalkar, “Paleo: A performance model for deep neural networks,” in *ICLR*, 2017.
[7] C. F. Rodrigues, G. Riley, and M. Luján, “Fine-grained energy profiling for deep convolutional neural networks on the jetson TX1, year=2017,” in *2017 IEEE International Symposium on Workload Characterization (IISWC)*.
[8] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, “Neuralpower: Predict and deploy energy-efficient convolutional neural networks,” in *Proceedings of The 9th Asian Conference on Machine Learning, ACML*, 2017.
[9] D. Stamoulis, E. Cai, D. Juan, and D. Marculescu, “Hyperpower: Power- and memory-constrained hyper-parameter optimization for neural networks,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 19–24.
[10] “Jetson AGX xavier developer kit,” accessed: 2018-06-30. [Online]. Available: developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
[11] “Harness AI at the edge with the jetson TX2 developer kit,” accessed: 2018-06-30. [Online]. Available: developer.nvidia.com/embedded/jetson-tx2-developer-kit
[12] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos, “Memory requirements for convolutional neural network hardware accelerators,” in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 111–121.
[13] J. D. Rodriguez, A. Perez, and J. A. Lozano, “Sensitivity analysis of k-fold cross validation in prediction error estimation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 569–575, 2009.
[14] N. Matloff, *Statistical regression and classification: from linear models to machine learning*. CRC Press, 2017.
[15] F. Murtagh, “Multilayer perceptrons for classification and regression,” *Neurocomputing*, vol. 2, no. 5-6, pp. 183–197, 1991.
[16] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.
[17] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX symposium on operating systems design and implementation OSDI 16)*, 2016, pp. 265–283.
[19] “Nvprof overview,” accessed: 2020-06-30. [Online]. Available: https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview
[20] E. Bisong, “Google colaboratory,” in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 59–64.

APPENDIX

A. Obtained results on Jeston TX2

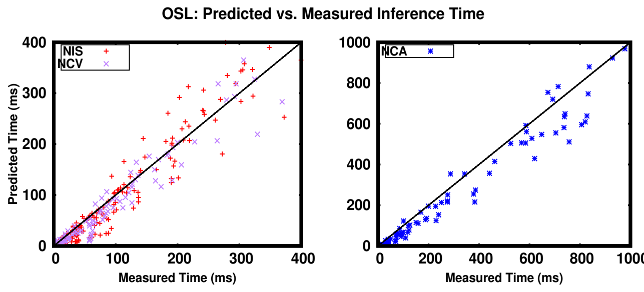

Fig. 11: Predicted vs. Measured Inference Time for OLS (TX2).

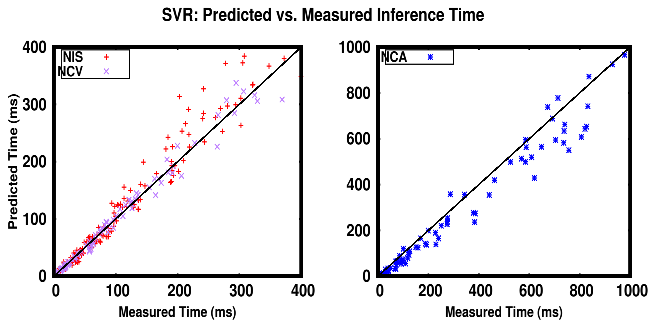

Fig. 12: Predicted vs. Measured Inference Time for SVR (TX2).

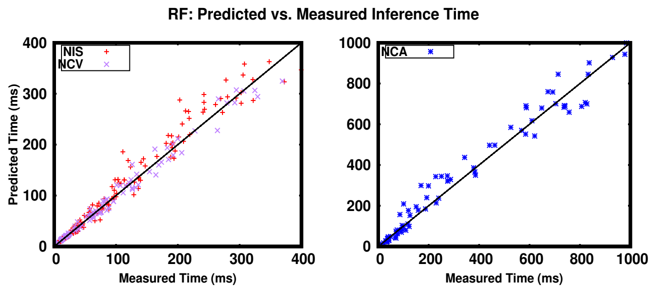

Fig. 13: Predicted vs. Measured Inference Time for RF (TX2).

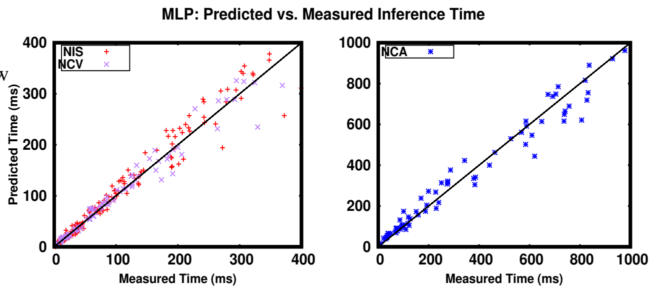

Fig. 14: Predicted vs. Measured Inference Time for MLP (TX2).

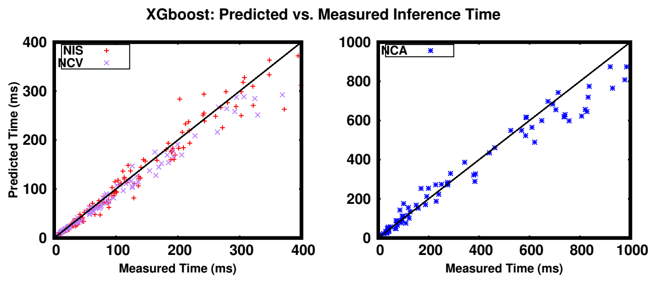

Fig. 15: Predicted vs. Measured Inference Time for XGBoost (TX2).