To Count or Not to Count: On the Usage of Performance Counter Metrics for Host to Embedded GPU Performance Estimation

Alexander Louis-Ferdinand Jung, Moritz Reiber, Jannik Steinmetz, Konstantin Lübeck, and Oliver Bringmann
Embedded Systems, Department of Computer Science
Eberhard Karls Universität Tübingen, Tübingen, Germany
{a.jung, moritz.reiber, jannik.steinmetz, konstantin.luebeck,
oliver.bringmann}@uni-tuebingen.de

Abstract

Artificial intelligence (AI) is undergoing a transition from cloud processing to local processing using embedded devices due to concerns regarding privacy, bandwidth limitations, and energy consumption. Consequently, AI models must be adapted to the constrained computing capabilities of embedded devices, such as embedded GPUs. This poses significant challenges, as the development and training of AI models still takes place on workstation or server GPUs with limited access to the hardware running the final AI function. Statistical performance estimation models offer insight into the runtime behavior of deep neural network (DNN) architectures. Furthermore, these estimation models can guide a neural architecture search (NAS) to prioritize the training of candidates that are compatible with the runtime constraints of the target hardware platform. However, implementing these statistical models requires either substantial training data or a certain degree of hardware knowledge to ensure the accuracy of the estimations. Benchmarking large quantities of training data is a time-consuming process, and knowledge about the hardware is not always available for commercial off-the-shelf (COTS) embedded GPUs.
To address these issues, we investigate the usage of hardware performance counter metrics collected on a host GPU as supplementary features employed by the performance estimator for the embedded GPU. In this paper, we examine different statistical performance estimation models and how the usage of hardware performance counter metrics influences estimation accuracy for both whole DNNs as well as single convolution layers.

1 Introduction

Today, artificial intelligence (AI) is being used in more and more fields and applications, like medical imaging, speech detection and processing, as well as highly automated driving. Many of those tasks happen in a compute- or power-constrained environment. This could be a video capsule for gastrointestinal endoscopy, a smart speaker, or an (electric) vehicle. Therefore, these AI tasks need to be developed for resource-constrained and/or power-efficient embedded devices. Behind most modern AI tasks are deep neural network (DNN) models, where a systematic neural architecture search (NAS) is superior to complex and error-prone manual design [1]. This is especially true when the desired network targets specific edge hardware platforms, which pose additional design questions and require the network designer to be both an expert in neural network design and hardware architectures. However, in order to perform a targeted search for specific hardware, a metric is needed to gauge the networks' runtime on that hardware. Use-cases for AI that require the application of edge hardware often coincide with strict runtime requirements, e.g., object detection in self-driving cars or the detection of an immediate need for therapeutic intervention in the medical setting. While the runtime of a DNN is independent of the specific input, e.g. different images of the same size, other parameters of the network, like e.g. the number of layers, input and output channels or kernel sizes can impact the runtime, as well as the accuracy, significantly. It is therefore crucial to perform the NAS with the runtime on the target hardware as an additional objective alongside accuracy. Direct measurements of candidate networks on the target hardware have several important disadvantages: they are time-consuming, require access to the hardware, and need an elaborate set-up and integration into the NAS process. Thus, especially for NAS, accurate performance estimation models are needed.

In the case of highly automated driving tasks, two example platforms are the NVIDIA Jetson AGX Xavier and Jetson AGX Orin. They combine an ARM CPU and an NVIDIA GPU, as well as the NVIDIA DLA (Deep Learning Accelerator). As these devices are COTS (commercial off-the-shelf) platforms, not much information is publicly available about the hardware architecture as well as the mapping of DNNs onto the hardware. Therefore, for those platforms, statistical performance models are a promising option. Creating such statistical performance models can be quite tedious, as one needs to collect many data points to achieve acceptable prediction errors.

While the information about the aforementioned platforms is limited, especially regarding the mapping of DNNs onto the hardware, NVIDIA provides tooling

to collect performance counter metrics. These metrics can be collected using specialized hardware counters during the execution of a program and include information like e.g., the number of executed instructions or the number of cache misses. These metrics can provide insight into the behavior and bottlenecks of an application but could also provide information that goes beyond metrics like the number of operations and layers in a DNN to a statistical performance estimator.

This paper proposes to use performance counter metrics gathered on a host GPU, together with global DNN features, like the number of layers, weights, and operations, to train statistical models to make an estimation of the runtime on the embedded GPU.

We make the following contributions: (1) We describe a process for selecting host GPU performance counter metrics based on their feature importance. (2) We then use these metrics to augment the features of three different statistical regression models for estimating the runtime of whole DNNs as well as single-layer convolutions on two embedded GPUs. (3) We further investigate how well the performance counter metrics selected for one embedded GPU are applicable to a different one and discuss the practical importance of the overhead associated with collecting hardware performance counter metrics when using such an estimation model during, e.g., a NAS.

2 Related Work

Zheng et al. [2] use performance counter features collected on two different host CPUs, like cache misses, branch misses, number of instructions, host cycles, and number of floating-point operations, to train different statistical models to estimate the runtime of benchmarks on an ARM processor. They evaluate their estimation models using reference timings from the default ARM CPU model of the gem5 simulator. They achieve an average prediction accuracy of 90 % using 15 700 program instances in their training set.

There are several publications covering the topic of statistical models for runtime estimation. Bouzidi et al. [3] use approximately 200 000 benchmarks of state-of-the-art DNNs and compare five different machine learning methods in terms of training time, hyperparameter tuning time, prediction latency, and runtime estimation accuracy. The authors use global features of the DNNs, like number of hidden layers, number of convolution layers, input size, number of weights, and overall number of operations, for training the estimators. They evaluate the models in three different task categories: (1) performance estimation of new image sizes, (2) performance estimation of new CNN variants, and (3) performance estimation of new CNN architectures, achieving mean absolute percentage errors (MAPE) between 7.67 % - 26.32 %.

The authors of ANNETTE [4] use single-layer and multi-layer benchmarks to make an estimation for an unknown DNN based on the contained layers while accounting for optimizations like layer fusion during the deployment on the target platform. They achieve a MAPE for an FPGA and an ASIC accelerator as low as 12.71 % for single-layer estimations and 3.47 % for whole networks while needing approximately 35 000 measurements per layer and platform.

A similar approach is used in nn-meter [5] where the authors perform up to 39 968 benchmarks for different execution kernels that contain typically fused layers. The trained random forest models achieve a root mean square percentage error (RMSPE) between 1.35 % - 22.25 % depending on the hardware platform and DNN combination.

Blackthorn [6] is not a statistical model, as the authors start with a set of step-wise functions from which they eliminate the ones that do not fit the measured data until only a single step-wise function remains. This is done for all required layer types. The selected step-wise functions are then used to estimate the runtime of the fused layers in a network. The predicted layer runtimes are accumulated and form the runtime estimation for the complete DNN. Their RMSPE for the prediction for the NVIDIA Jetson Nano and TX2 platforms ranges between 5.89 % - 6.10 % for single Conv2D layers.

Jung et al. [7] exploit the regular runtime behavior of hardware AI accelerator platforms, like the UltraTrail [8] and VTA [9] accelerators, as well as the NVIDIA Jetson AGX Xavier GPU, reducing the overall number of required training points by only benchmarking so-called 'Performance Representatives (PRs)' and mapping layer parameter configurations to these PRs before making an estimation. This results in an estimation error between 7.09 % - 13.13 % for convolution layer estimations and 2.9 % - 19.6 % for whole DNN estimations while needing significantly less training data compared to other state-of-the-art approaches. This result highlights that it can be beneficial to incorporate further information, like the regular runtime behavior of a hardware platform, into the benchmarking and estimation approach.

3 Host to Embedded GPU Performance Estimation

When looking at the related work, one quickly notices that the different methods all achieve similar estimation accuracies but vastly differ in terms of the number of training benchmarks required. Only Jung et al. [7] presented a concrete approach on how to reduce the number of training samples while maintaining estimation accuracy. Unfortunately, their approach requires extensive effort when it comes to the so-called black-box architectures, like the NVIDIA Jetson AGX Xavier GPU: In a first step, sweep benchmarks have to be performed to determine the PRs, then benchmarks of all required layers have to be measured and used for building the single-layer models. Finally, multi-layer benchmarks need to be performed to determine the so-

called fusing factor, which is needed for a performance estimate of a whole DNN.

Therefore, we want to combine the end-to-end estimation approach for whole DNNs by Bouzidi et al. [3] with the idea of Zheng et al. [2] of using hardware performance counter metrics to introduce information about the hardware behavior of black-box architectures into the model without the need of explicitly determining this behavior first with sweep benchmarks. Consequently, the target devices for which we want to build performance estimators in this work are NVIDIA embedded GPUs as found in the NVIDIA Jetson Xavier and Orin devices. The reason for choosing these platforms is twofold. First, the NVIDIA Jetson platforms are widely used in industry [10] and research [3, 6, 11, 7]. Second, NVIDIA desktop and server GPUs are typically used in DNN development and training. Thus, the required performance counter metrics can be collected on the host during DNN development/training and automatically be used to estimate the runtime of the DNN on the embedded GPU, even if there is no access to the embedded GPU.

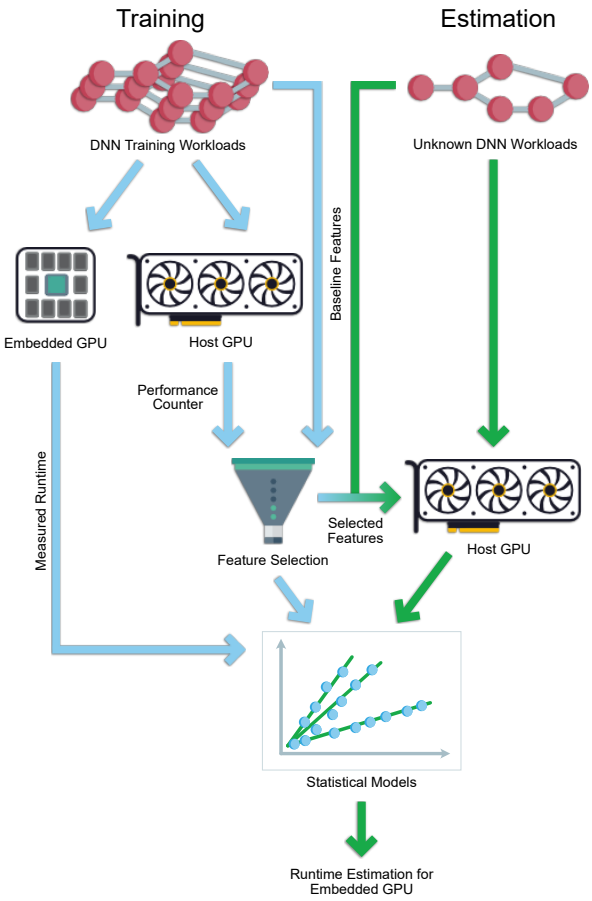

Figure 1: Flow of the host to embedded GPU runtime estimation using performance counters.

3.1 Whole DNN Runtime Estimation

When building statistical models for whole DNN estimation, the question arises what features to use for training the models. As DNNs consist of many layers with different parameters, the parameters of the single-layers cannot be used directly. Inspired by [3], we use the global features listed in the left column of Table 1 as our baseline features. The training set comes from the nn-meter [5] benchmark dataset available at their GitHub repository [12] from which we randomly sampled 500 instances of each DNN architecture contained in the dataset, resulting in a total of 5000 DNNs.

Whole DNN Features	Single-Layer Conv Features
#Conv-Layers	#input channels C
#Pointwise Conv-Layers	input height C_h
#Depthwise Conv-Layers	input width C_w
#Fully-Connected-Layers	#output channels K
#Activation-Layers (e.g. ReLU, Clip, etc.)	kernel height F_h
#Elementwise-Layers (e.g. Add, Mul, etc.)	kernel width F_w
	stride s
#operations per layer	padding pad
#inputs per layer	#inputs
#outputs per layer	#outputs
#weights per layer	#operations
#operations total	#weights

Table 1: Baseline features of whole DNNs and single-layer convolutions. The *#-symbol* means 'number of' and *per layer* means 'for every layer type listed in the first part of the whole DNN features'.

We collected the runtime on the target embedded GPUs and used the baseline features to train baseline models, using Random Forest, Gradient Boosting, and XGBoost as our estimators. A comparison of different statistical models was deemed necessary because Bouzidi et al. [3] observed varying performance of different regression methods for their different task categories. We then augment these baseline features with host performance counter metrics measured on a host GPU.

The number of measurable performance counter metrics is massive, and it takes more time to collect more metrics. Consequently, we wanted to reduce the amount of metrics that will be used as a feature in the estimation model to reduce the profiling time and also mitigate overfitting and reduce training time of the estimator. Therefore, we used a feature selection approach called *recursive feature addition* from the Python library Feature-engine [13]. It first trains an estimator using all the available features. It then orders the features by their feature importance, trains an estimator using only the most important feature, and stores its performance. Lastly, it iteratively adds more and more features (in order of their importance) until the estimator's performance doesn't increase enough anymore. This feature selection was done five times for each statistical model type individually, as some features might be more important for certain types of models than for others, and the union of features of

all five runs was used to improve robustness. With these selected features, we trained our estimators using different training sample sizes to determine how many samples are needed for an accurate runtime estimation.

To make an estimation for an unknown DNN with the trained model, one needs to collect the performance counter metrics determined by the feature selection during the training process and give the feature vector to the statistical model to obtain an estimate for the runtime of that DNN on the target hardware. The flow of the training and estimation process for whole DNNs is depicted in Figure 1.

3.2 Single-Layer Runtime Estimation

While the main focus of this work is to build performance estimators for whole DNNs, an extension of our approach to single-layers enables a more exhaustive evaluation and comparison to the state-of-the-art. Therefore, we additionally benchmarked and modeled single-layer 2D convolutions with the following parameters:

$$Conv2D(C, C_h, C_w, K, F_h, F_w, s, pad) \quad (1)$$

with input channels C, input height C_h and width C_w, output channels K, filter height F_h and width F_w, stride s and padding *pad*. These layer parameters, together with metrics computed from these parameters, will be used as baseline features (see right column of Table 1) in the statistical models, as other studies like [4, 7] have also used similar features.

The process for building and using statistical models for the single-layer estimation looks very similar to the one depicted in Figure 1 for whole DNNs. We used a Random Forest regression model and also performed a recursive feature addition for the host performance counter metrics to reduce the number of features in the final feature vector. Random Forest regression was chosen because other works have already shown this type of statistical model to be quite capable of estimating single-layer runtime [4, 7], which we could empirically verify with initial testing.

4 Evaluation

4.1 Experimental Setup

The runtime was measured on the NVIDIA Jetson AGX Xavier GPU, running CUDA 11.4 and TensorRT 8.5.2, and the Jetson AGX Orin GPU, running CUDA 12.2 and TensorRT 8.6.2. The measurements were performed using the TensorRT backend of Apache TVM [14] (version 0.15.0-dev0).

The tool to collect performance counters for NVIDIA GPUs is called NVIDIA NSight Compute (NCU). It offers many different performance metrics for different parts of the GPU. From the caches, over streaming multiprocessors, CUDA cores, memory accesses, etc., to the overall execution time of the entire executed program. Some of the reported metrics are ratios of other metrics. As their information should be contained in the metrics from which the ratio is computed, we did not collect those ratio metrics. Consequently, we chose to only collect metrics that have one of the following units: `byte`, `cycle`, `ns`, `inst` or are a quantity, like, e.g., the number of L1 cache data bank reads. The feature selection described in Section 3.1 was performed on these collected metrics. For a complete list of the measurable metrics, refer to the official documentation [15].

We integrated the NCU command-line tool into our benchmarking framework to be able to automatically collect the performance counter metrics of whole DNN and single-layer benchmarks on a host GPU. Before profiling the execution, it is essential to conduct a warm-up run in order to capture the metrics of only the runtime behavior. The performance counter metrics were collected on an NVIDIA GeForce RTX 2080 Ti installed in a workstation computer (AMD Ryzen 5 3600 CPU, 48 GB RAM) running Ubuntu 22.04.

4.2 In-distribution Runtime Estimation

Initially, we will look at the performance of the runtime estimation when estimating DNNs and single-layer convolutions inside the distribution of the training data. Obviously, the test data cannot be part of the training data. In-distribution means that the test set comes from the same statistical distribution as the training set [16]. Subsequently, we will also look at the performance of the runtime estimation when estimating DNNs and single-layer convolutions that come from a different statistical distribution (see Section 4.3).

4.2.1 Whole DNNs

For the in-distribution evaluation, we first split the complete dataset containing 5000 DNNs into a *test set* of size 1000 containing a randomly selected 20 % of the complete dataset. This *test set* is fixed and only used for evaluation. The remaining 80 % of the dataset forms the *training set*. The five feature selection runs (see Section 3.1) were performed on the complete *training set*. In Table 2 the selected features for the whole DNN Random Forest estimation model for the Xavier GPU are listed as an example. The different training dataset sizes were selected from the complete *training set* using different random seeds.

Figure 2 shows the MAPE of the three different statistical models: Random Forest Regressor (RF), Gradient Boosting Regressor (GB) from [17] and XGBoost Regressor (XGB) from [18]. For all models, using the baseline features performs worse than when using the features from the feature selection, but the difference is rather small for RF and XGB and a bit bigger for GB. For an in-distribution estimation, the baseline features seem to be enough to capture the runtime behavior.

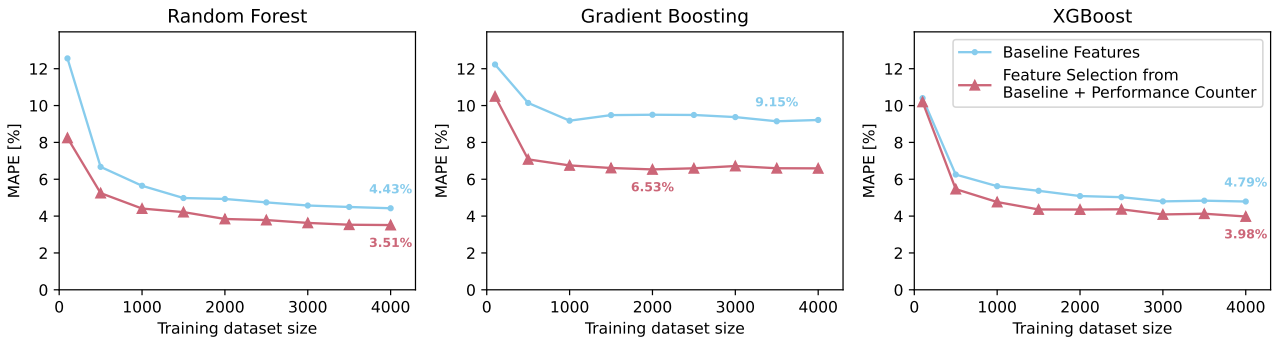

Figure 2: MAPE of different statistical models for the in-distribution test set on the NVIDIA Jetson AGX Xavier GPU. The best MAPE value for each combination of estimation model, feature set, and training dataset size is labeled. The lines between the data points serve to improve visibility and help to distinguish the evaluated feature sets and do not indicate interpolation.

Selected Features
lts__cycles_in_region.sum (cycle)
sys__cycles_in_frame.sum (cycle)
total_ops
smsp__inst_executed_pipe_uniform.sum (inst)
smsp__sass_thread_inst_executed_ops_fadd_fmul_ffma_pred_on.sum (inst)
fbpa__dram_cycles_elapsed.sum (cycle)
l1tex__t_set_accesses_pipe_lsu_mem_global_op_st.sum ()
smsp__mio2rf_writeback_active_pipe_lsu.sum (cycle)
gpu__time_duration_measured_wallclock.sum (ns)
smsp__inst_executed_op_shared_st_pred_off_all.sum (inst)
smsp__inst_executed_pipe_adu.sum (inst)
smsp__issue_inst0.sum (cycle)
sm__cycles_active_shader_cs.sum (cycle)
sm__sass_thread_inst_executed_op_fp32_pred_on.sum (inst)
smsp__pipe_fma_cycles_active.sum (cycle)
smsp__sass_inst_executed_op_ld.sum (inst)
fbpa__cycles_active.sum (cycle)
smsp__thread_inst_executed_pipe_fma_pred_on.sum (inst)
act_num_layers
smsp__sass_thread_inst_executed_op_ffma_pred_on.sum (inst)
conv_num_ops

Table 2: Features that were selected by the feature selection process from the measured performance counters and the baseline features for the whole DNN Random Forest model for the NVIDIA Jetson AGX Xavier GPU.

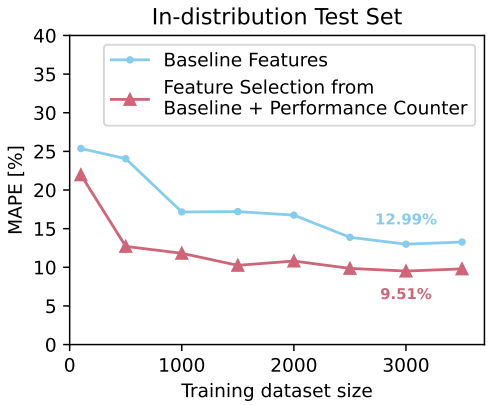

Figure 3: MAPE comparison for the in-distribution single Conv2D layers on the NVIDIA Jetson AGX Xavier GPU.

4.2.2 Single-Layer Convolutions

For the in-distribution evaluation of the single-layer estimation, we measured runtime and performance counter metrics of 4430 (20 % test set of size 886) Conv2D layers with the parameters (see Section 3.2) randomly selected within a reasonable range. We trained RF regression models on randomly sampled datasets of different sizes using the features from the right column of Table 1 as well as features selected via the feature selection described in Section 3.1.

Figure 3 shows the MAPE of the runtime estimation for the Xavier GPU for the single-layer Conv2D. The model using the performance counter features again performs slightly better than the baseline features. Interestingly, the overall estimation error is significantly worse than for the whole DNN estimation. This could be due to the short runtime of single-layer convolutions compared to whole DNNs resulting in less reliable performance counter measurements on the host GPU.

4.3 Out-of-distribution Runtime Estimation

When training a statistical model, one should not only look at the in-distribution estimation performance but also at the out-of-distribution performance. The latter provides insight into how well the estimation model generalizes, which is essential if an estimation model is to be used to estimate the runtime of DNNs different from the ones that were benchmarked during model creation.

4.3.1 Keras Modelzoo DNNs

For the out-of-distribution evaluation, we used 15 state-of-the-art DNNs from the Keras modelzoo [19]: DenseNet121, DenseNet169, MobileNet, MobileNetV2, ResNet101, ResNet101v2, ResNet152, ResNet152v2, ResNet50, ResNet50v2, ResNetRS101, ResNetRS50, VGG16, VGG19, and Xception. While many of the general architectures of these DNNs are also part of the training dataset these specific models are considered out-of-distribution because first, the ONNX files of the same architectures show some differences in regard to

e.g. the used activation functions, pooling types and pooling windows and the presence of reshape, squeeze, padding and softmax operations. Moreover, the larger variants like e.g. DenseNet169, ResNet101(v2) and ResNet152(v2) are not present in the nn-meter dataset. This can also be seen in Figure 4 which shows the distribution of the number of specific layers in the two datasets. One can clearly see that while there is naturally some overlap between the number of layers, the DNNs from the Keras modelzoo tend to have more layers on average as well as candidates outside the maximum values seen in the nn-meter dataset.

Figure 5 shows the MAPE values of the three evaluated statistical models for different training dataset sizes. This time, using the features selected from the performance counter metrics and the baseline features outperforms the estimation performance of only the baseline features considerably, but the overall performance is worse than for the in-distribution estimation.

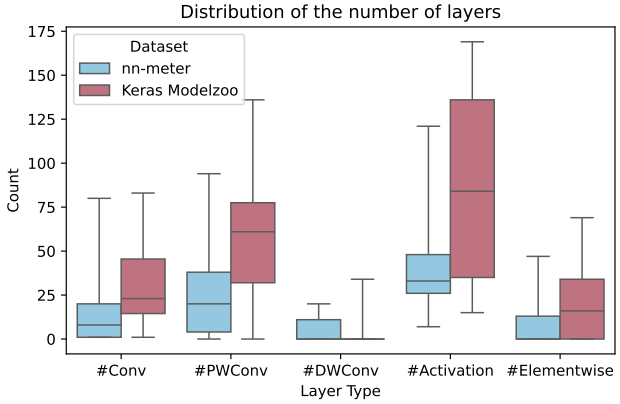

Figure 4: Comparison of the distribution of the number of layers per types across the nn-meter dataset and the Keras modelzoo.

4.3.2 Keras Modelzoo Single-Layer Convolutions

Similarly, for the out-of-distribution evaluation of the single-layer estimation models, we extracted the parameters of all Conv2D layers contained in the Keras modelzoo models, estimated their runtime, and compared the estimations with the measured ground truth. Figure 6 shows that the overall performance on the out-of-distribution test set is worse than the in-distribution performance. This is similar to what we observed for the in- and out-of-distribution estimation of whole DNNs. But the estimation error for out-of-distribution single-layer convolutions is comparable to the estimation error for the out-of-distribution whole DNN estimation presented in Figure 5.

Generally, when estimating the out-of-distribution test sets, using the feature sets augmented with performance counters resulted in significantly lower MAPE values compared to the baseline features for both whole DNN and single-layer Conv2D estimations.

4.4 Estimation for a Different Hardware Platform

The advantage of the presented approach, using host GPU performance counter metrics to augment the baseline features for whole DNN estimation, is that such metrics would only need to be collected once and could then be used to train runtime estimation models for different target GPUs. As described in Section 3.1 it is impractical to always collect all the available performance counter metrics. Therefore, it would be ideal if the metrics selected by the feature selection algorithm for one embedded GPU could also be used for the estimation for a different embedded GPU.

To evaluate this, we used the features selected for the estimation of the runtime on the NVIDIA Jetson AGX Xavier GPU to train statistical estimation models for the NVIDIA Jetson AGX Orin GPU and compared the results to the estimation error when performing a feature selection for the Orin GPU itself. Figure 7 shows the in-distribution whole DNN runtime estimation MAPE using the baseline features, the features from the Xavier feature selection, and the Orin feature selection. For the RF and GB models, the baseline features outperform the performance counter features by a tiny margin, while the features selected for the Xavier GPU result in the best MAPE when using the XGB model. But as all the results are very close in terms of MAPE, all feature sets seem to be capturing the relevant information equally for all model types.

The out-of-distribution whole DNN estimation results are shown in Figure 8. For the RF model, the feature set selected specifically for the Orin GPU performs considerably better than the baseline features and the features selected for the Xavier GPU. For the GB model, all feature sets perform similarly, and for the XGB model, the features selected for the Xavier and Orin GPU both perform equally better than the baseline features. While the best result for the Orin GPU was achieved with the RF model using the Orin GPU feature selection, the error when using the XGB model with the features selected for the Xavier GPU lies within a couple of percentage points. Consequently, the features selected for the estimation of the Xavier GPU are to some extent (depending on the regression model) applicable to the estimation of the Orin GPU and could be reused in a real-world scenario.

4.5 Cost of Measuring Performance Counter Metrics

For all evaluated models, for both in-distribution and out-of-distribution estimation, and on all hardware platforms, it can be observed that the MAPE decreases the most when increasing the training dataset from 100 to 1000, but after that, the improvements in estimation performance decline significantly. This is especially interesting in light of the 'cost' associated with collecting the performance counter metrics.

Figure 9 shows the time it took to measure the run-

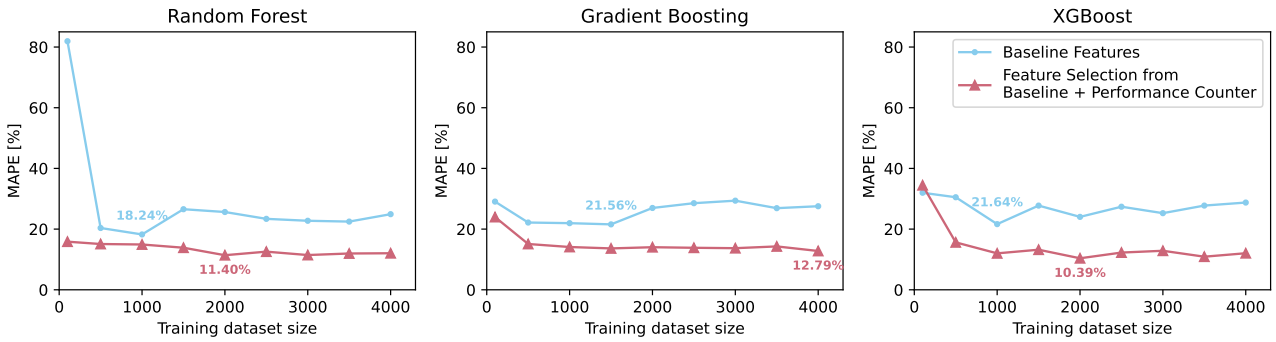

Figure 5: MAPE of different statistical models for the out-of-distribution test set on the NVIDIA Jetson AGX Xavier GPU. The best MAPE value for each combination of estimation model, feature set and training dataset size is labeled.

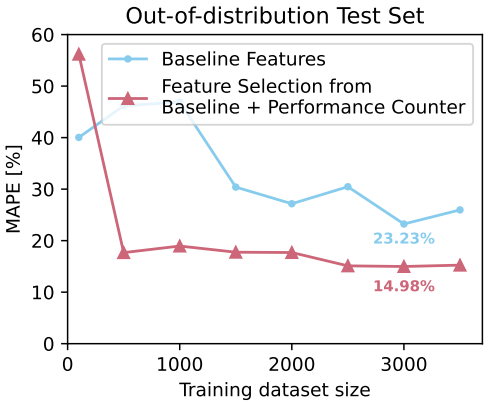

Figure 6: MAPE comparison for the out-of-distribution single Conv2D layers on the NVIDIA Jetson AGX Xavier GPU.

time of the out-of-distribution DNNs on the NVIDIA Jetson AGX Xavier GPU compared to the time it took to profile only those performance counter metrics that were selected by the feature selection described in Section 3.1 for the whole DNN RF model for the Xavier GPU. For the runtime measurements on the Xavier GPU, the measurement was repeated 50 times to mitigate variances when performing runtime measurements on real hardware. The profiling tool NVIDIA NSight Compute uses various techniques to achieve reproducible results [20]. Therefore, the profiling was only performed once for each DNN. For most of the DNNs in Figure 9, the difference between the measurement time on the target hardware vs. the profiling time on the host GPU is relatively minor, and at least always in the same order of magnitude.

There are two aspects of 'costs' associated with a performance estimation model. First, the time it takes to collect the training data and, second, the time it takes to make an estimation for an unknown DNN. During the data collection phase of the model building, the steps of profiling performance counter metrics on the host GPU and measuring the runtime on the embedded GPU could be parallelized, since the measurement and profiling times are similar, such that no additional time overhead is introduced during training data collection. Hence, requiring fewer benchmark points when using performance counter metrics compared to using only the baseline features while achieving competitive estimation errors results in shorter model building times.

When model training has finished and it is being used for estimating an unknown DNN's runtime, the time it takes to capture the feature vector comes into play. On the one hand, as stated before, profiling performance counter metrics takes a similar amount of time as measuring on the target platform. So if the target platform is available, the real-world measurement avoids the error associated with any estimation model. On the other hand, the target hardware might not be available all the time, if at all, and, moreover, automatically measuring requires an elaborate set-up and integration into, e.g., the NAS process. Looking at the NAS use-case for performance estimation models, we have a big advantage when it comes to the cost of profiling the required performance counters. While not every architecture candidate will be fully trained to gauge its accuracy during the NAS, as this is usually too costly, a lot of NAS approaches do perform either a proxy training on a limited training set or with fewer epochs [21, 22, 1] or carry out some forward passes, e.g., to compute zero-cost metrics [23, 24]. As this is done on the host GPU, the performance counters can be collected during these passes and then be used in the estimator for the targeted black-box embedded GPU. As the main purpose of the execution of the network on the host GPU is inherent to the NAS algorithm (i.e., accuracy estimation), the collection of performance counters incurs a negligible overhead.

4.6 Comparison with State-of-the-Art

Table 3 shows a comparison to other state-of-the-art (SOTA) runtime estimation approaches. It shows, the best literature reported MAPE and/or RMSPE values and training dataset sizes of approaches that were evaluated for NVIDIA embedded GPUs.

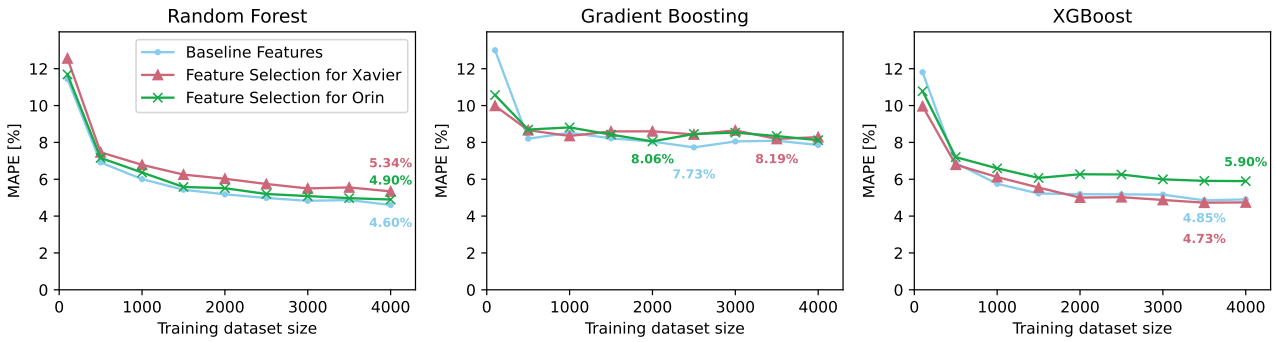

Figure 7: MAPE comparison for the in-distribution whole DNNs on the NVIDIA Jetson AGX Orin GPU. Note that the red curve shows the MAPE of the estimation for the Orin GPU using the features previously selected for the Xavier GPU, and the green curve shows the MAPE when using the features selected specifically for the Orin GPU.

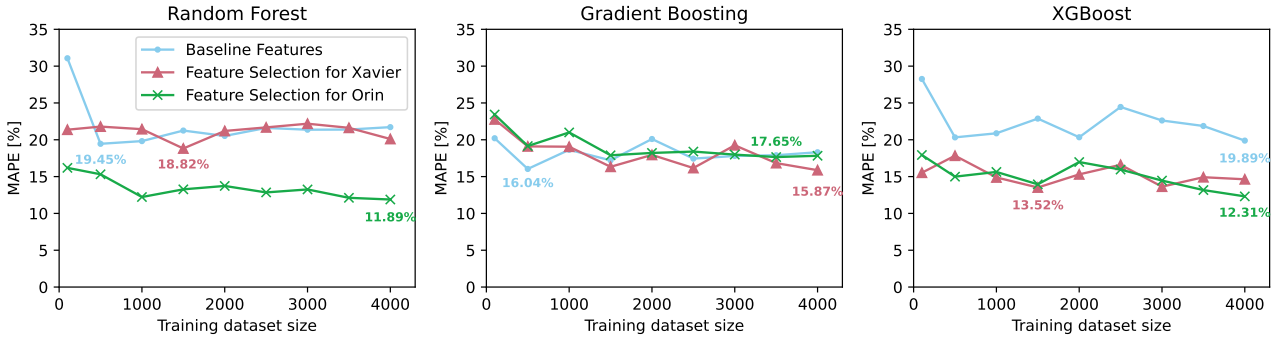

Figure 8: MAPE comparison for the out-of-distribution whole DNNs on the NVIDIA Jetson AGX Orin GPU. Note that the red curve shows the MAPE of the estimation for the Orin GPU using the features previously selected for the Xavier GPU, and the green curve shows the MAPE when using the features selected specifically for the Orin GPU.

Note that while Blackthorn [6] shows very good estimation performance for whole DNNs, it is not a statistical model. Moreover, the authors state that the whole DNN error being lower than the error of the single-layer estimations is due to these errors compensating each other when summing up the individual estimations. This could also lead to a significantly larger error, depending on the layers contained in the DNN to be estimated. Moreover, we have observed that the NVIDIA Jetson AGX Xavier and Orin GPUs experience a sophisticated fusing behavior that could be difficult to capture using fixed step-wise functions.

Bouzidi et al. [3] is the only performance estimation model that uses an end-to-end approach, where the estimation is done directly for a DNN and not via a combination of single-layer estimations. While our MAPE values on the out-of-distribution test set are a bit higher than their best results, it is important to note that they used ∼200 000 DNNs for training their estimators, while our best results have been achieved using training dataset sizes of 2000 and 4000 DNNs for the AGX Xavier and AGX Orin GPUs, respectively.

Wess et al. [11] also achieve a lower estimation error compared to our approach, but their method requires additional measurements to build a look-up table with which the final single-layer measurements will be corrected to account for data transfers that are not part of the execution if the layer is contained in a whole DNN.

The approach of Jung et al. [7] seems to accurately estimate the runtime for the so-called white-box and gray-box accelerators but struggles with the black-box nature of the NVIDIA Jetson AGX Xavier GPU.

While our approach also does not outperform the state-of-the-art estimation results for single Conv2D layers, we achieve comparable results while needing far fewer training benchmarks.

5 Conclusion

This study introduced the usage of host GPU performance counter metrics as features in statistical models for the runtime estimation on an embedded GPU, as found in the NVIDIA Jetson AGX Xavier and Orin devices. For the in-distribution estimation, the augmented feature set outperforms the baseline network or layer-level featurs only by a small margin. For the out-of-distribution DNNs and single Conv2D layers, the performance counter metrics provide the statistical models with relevant information, leading to a significant improvement in MAPE.

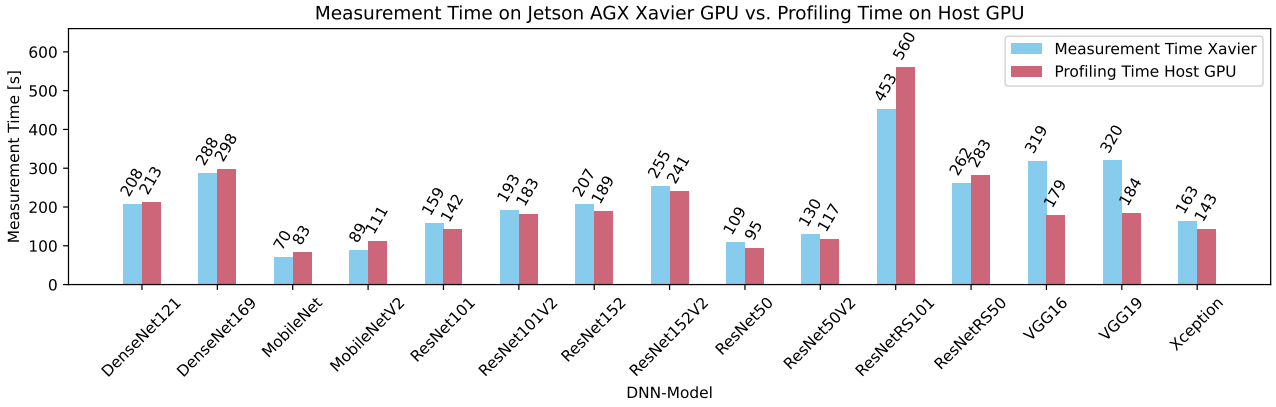

Figure 9: Comparison of the time it took to measure the runtime on the NVIDIA Jetson AGX Xavier GPU and the time it took to profile the performance counter metrics, that were selected for the whole DNN Random Forest model for the Xavier GPU, on the host GPU (including the warm-up run).

Paper	Type	NVIDIA Jetson GPU	Dataset Size	MAPE	RMSPE	Model Type
Blackthorn [6]	DNN	Nano TX2	Combination of single-layer models	2.95 %[1] 4.29 %[1]	1.71 % 3.06 %	Step-wise functions selected through measurements
	Conv2D	Nano TX2	15 000 per platform	- -	5.89 % 6.10 %	
Bouzidi et al. [3]	DNN	AGX Xavier TX2	∼200 000 DNNs per platform	7.67 % 8.37 %	- -	SVR[2]
Wess et al. [11]	DNN	AGX Xavier	∼7000[3]	6.90 %	-	RF[4]
Jung et al. [7]	DNN	AGX Xavier	up to 9000[5]	19.60 %	20.17 %	RF[4] + linear model
	Conv2D	AGX Xavier	8000	13.13 %	27.06 %	RF[4]
Performance Counter	DNN	AGX Xavier AGX Orin	2000 DNNs 4000 DNNs	10.39 % 11.89 %	12.29 % 14.62 %	XGB[6] RF[4]
	Conv2D	AGX Xavier	3000	14.98 %	20.36 %	RF[4]

Table 3: Comparison of the estimation MAPE and RMSPE with different SOTA methods.
[1]MAPE values were computed from the reported times, [2]Support Vector Regression, [3]∼1000 for each of the 7 layer types measured + measurements for correction LUT, [4]Random Forest Regression, [5]per layer type + up to 500 multi-layer benchmarks per building-block, [6]XGBoost Regression

While other state-of-the-art performance estimation approaches often yield a smaller prediction error, this often comes with an increased amount of training data or manual effort when building the estimation model. Consequently, if the DNN needs to be executed on a fast host GPU anyway during either manual model development or a NAS, the overhead of collecting the host performance counter metrics becomes negligible. Unfortunately, the presented approach is limited to embedded GPUs where a somewhat similar host GPU exists and that provides the ability to collect hardware performance counter metrics.

Interestingly, the comparison between the features selected for the Xavier and Orin GPU in terms of estimation error for the Orin GPU revealed that it appears to be dependent on the regression model type whether the selected features are applicable to another embedded GPU. It would be intersting to explore the differences between the hardware platforms, the regression models and the feature sets more in depth in future work. Further research could also evaluate the estimation performance of our models trained for DNNs on other architectures, such as large language models, refine the feature selection process, or try to extract relevant information from, e.g., the deployment tools and use this information to augment the features of a statistical performance estimation model, removing the need to execute the DNN on the host altogether when making an estimation for the target platform.

6 Acknowledgement

This work has been funded by the German Federal Ministry of Research, Technology and Space (BMFTR) under grant number 01IS22086H (MANNHEIM-FlexKI).

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[2] X. Zheng, P. Ravikumar, L. K. John, and A. Gerstlauer, "Learning-Based Analytical Cross-Platform Performance Prediction," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, (Samos, Greece), pp. 52–59, IEEE, July 2015.

[3] H. Bouzidi, H. Ouarnoughi, S. Niar, and A. A. E. Cadi, "Performance Prediction for Convolutional Neural Networks on Edge GPUs," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, CF '21, ACM, May 2021.

[4] M. Wess, M. Ivanov, C. Unger, A. Nookala, A. Wendt, and A. Jantsch, "ANNETTE: Accurate Neural Network Execution Time Estimation with Stacked Models," *IEEE Access*, vol. 9, pp. 3545–3556, 2021.

[5] L. L. Zhang et al., "nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '21, ACM, June 2021.

[6] M. Lechner and A. Jantsch, "Blackthorn: Latency Estimation Framework for CNNs on Embedded Nvidia Platforms," *IEEE Access*, vol. 9, p. 110074–110084, 2021.

[7] A. L.-F. Jung, J. Steinmetz, J. Gietz, K. Lübeck, and O. Bringmann, *It's All About PR: Smart Benchmarking AI Accelerators Using Performance Representatives*, p. 59–75. Springer Nature Switzerland, 2025.

[8] P. P. Bernardo, C. Gerum, A. Frischknecht, K. Lübeck, and O. Bringmann, "UltraTrail: A Configurable Ultralow-Power TC-ResNet AI Accelerator for Efficient Keyword Spotting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, p. 4240–4251, Nov. 2020.

[9] T. Moreau, T. Chen, L. Vega, J. Roesch, E. Yan, L. Zheng, J. Fromm, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "A Hardware-Software Blueprint for Flexible Deep Learning Specialization," 2019.

[10] "NVIDIA DRIVE Partner Ecosystem." `https://www.nvidia.com/en-us/solutions/autonomous-vehicles/partners/`. Last accessed: 2025-11-28.

[11] M. Wess, D. Schnöll, D. Dallinger, M. Bittner, and A. Jantsch, "Conformal Prediction Based Confidence for Latency Estimation of DNN Accelerators: A Black-Box Approach," *IEEE Access*, vol. 12, p. 109847–109860, 2024.

[12] "nn-Meter Benchmark Dataset." `https://github.com/microsoft/nn-Meter?tab=readme-ov-file#benchmark-dataset`. Last accessed: 2026-01-27.

[13] S. Galli, "Feature-engine: A Python package for feature engineering for machine learning," *Journal of Open Source Software*, vol. 6, no. 65, p. 3642, 2021.

[14] T. Chen et al., "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," 2018.

[15] "NVIDIA NSight Compute Profiling Guide, Metrics Guide." `https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#metrics-guide`. Last accessed: 2025-11-28.

[16] Setlur, Amrith and Li, Oscar and Smith, Virginia, "Two Sides of Meta-Learning Evaluation: In vs. Out of Distribution," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 3770–3783, Curran Associates, Inc., 2021.

[17] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] 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*, KDD '16, (New York, NY, USA), p. 785–794, Association for Computing Machinery, 2016.

[19] "Keras Applications." `https://keras.io/api/applications/`. Last accessed: 2025-12-05.

[20] "NVIDIA NSight Compute Profiling Guide, Reproducibility." `https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html#reproducibility`. Last accessed: 2026-01-29.

[21] B. Moser, F. Raue, J. Hees, and A. Dengel, "Less is More: Proxy Datasets in NAS approaches," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, p. 1952–1960, IEEE, June 2022.

[22] M. Sheehan and O. Yakimenko, "Neural architecture search applying optimal stopping theory," *Frontiers in Artificial Intelligence*, vol. 8, p. 1643088, 2025.

[23] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," *arXiv preprint arXiv:2101.08134*, 2021.

[24] J. Lukasik, M. Moeller, and M. Keuper, "An evaluation of zero-cost proxies-from neural architecture performance prediction to model robustness," *International Journal of Computer Vision*, vol. 133, no. 5, pp. 2635–2652, 2025.