3

Prediction of Neural Network Latency on Embedded GPU Accelerators

Adrian Osterwind[1], Domenik Helms[1], and Verena Klös[2]

[1]German Aerospace Center (DLR), Germany
[2]Carl von Ossietzky Universität Oldenburg, Germany

Abstract

AI systems are transforming many application areas, but resource constraints prevent many especially time critical embedded applications. In particular, a fast prediction of expected latency and energy costs can enable the hardware-aware design of such systems. Thus, we started developing a latency model for the NVIDIA Jetson platform, which, with its GPU-based architecture is interesting in e.g. automotive vision use-cases. Using latency models in automatic optimization flows yields benefits, compared to abstract metrics like parameter count. However, it needs to be fast, for optimization algorithms to cover large search spaces. We developed a layer-wise model using the parameters of the layer (like input size) and a number of hardware measurements to produce a model of the network latency. On the Jetson platform the characterization yielded a model, that in first tests provided three key insights: Using only a few measurements we can already accurately predict upper and lower bound of the execution time. Using more information, we can, under ideal conditions, estimate latency with a mean average percentage error below 3%. Lastly, we found that sometimes a small change of input parameters can lead to up to 10% reduction in latency for that layer.

Keywords: embedded AI, resource modelling, AI, embedded systems.

3.1 Introduction

With the increase in competency of neural networks, they have seen deployment in more and more critical applications such as automotive. In this sector, the push for fully autonomous driving demands increasingly complex functions to be executed on the onboard computer.

To be able to execute these functions together on an embedded system while adhering to critical time boundaries, they must be optimized. These optimizations require knowledge about hardware behaviour to judge the changes in performance.

This is often obtained using either high-level metrics such as the number of floating-point operations (FLOPs) or weights in a neural network, or by conducting experiments with hardware-in-the-loop.

In this paper, we explore a latency model for a GPU-based system, such as those used in automotive use-cases, which promises an accurate model of the hardware with a minimal number of measurements. This work in progress is tested on the example of an NVIDIA Jetson development board, which is a relevant hardware platform for autonomous vehicles.

The remainder of this document is structured as follows. In Section 3.2, we discuss previous solutions to the modelling problem for AI hardware accelerators, which will inform some decisions in our approach. Based on the knowledge gained, we develop our own approach in Section 3.3, discussing the general hardware model and how to characterize it effectively. Initial results of our experimentation are shown in Section 3.4, showing the hardware behaviour under changing layer configurations and the first modelling approaches. In Section 3.5, we summarize our findings and discuss next steps.

3.2 Related Work

The need for performance prediction has been recognized by many researchers and embedded AI engineers in the past.

Approaches range from very high-level metrics such as number of floating-point operations (FLOPs) [1], to low-level simulations of the hardware [2]. Both are valid for certain use-cases, however, they come with some inherent issues. Using a metric such as FLOPs results in a very simple and fast model of hardware performance. However, they might not accurately account for behaviour such as overhead for memory management and hardware optimizations [3].

Low-level simulations, in contrast, capture these behaviours very well and can thus be much more accurate. The cost of accuracy is the need for in-depth knowledge of the hardware, which might not be available on proprietary systems [4]. Furthermore, the complexity of these simulations results in high simulation overheads [2, 5].

A compromise between these two extremes is a hardware-aware black box method.

For this work, we are interested in a hardware-aware black box method. Previously, we presented a model for the Neural Compute Stick 2 in [6]. To achieve this, we used a generalized model of the behaviour of neural network layers, which is tailored to the hardware by latency measurements. For this, we used a simple linear regression between points. To characterize the model, we defined a search space to interpolate within. In this search space, we selected a random set of samples, which needed to be dense, to cover most non-linearities.

The work in [7] tackled this issue by using random forest regression. However, this still requires many datapoints to characterize a hardware platform.

By trying to determine points of interest for characterization, [8] reduces this number. These points of interest are determined in a sweep of the target parameter of the neural network layer.

In this work, we will use a similar methodology, sweeping through parameter ranges to determine points of interest. However, we will use composite functions for each dimension, which promise to capture more details gained in the sweep, allowing us to reduce the number of measurements taken.

3.3 Approach

For our modelling approach we exploit the fact that the latency of a network can be closely approximated by summing up the latencies of all layers. This way, the prediction can be done layer by layer. Furthermore, we assume a certain regularity in the change of latency of a layer.

The prediction for a layer depends on its type, e.g., Conv2D and parameters, e.g., number of channels or filters (see Figure 3.1), as well as some hardware-specific numbers that we model as variables in the general hardware-agnostic model. By instantiating the variables with values obtained from measurements, we get a hardware-specific model, that can be used to derive predictions.

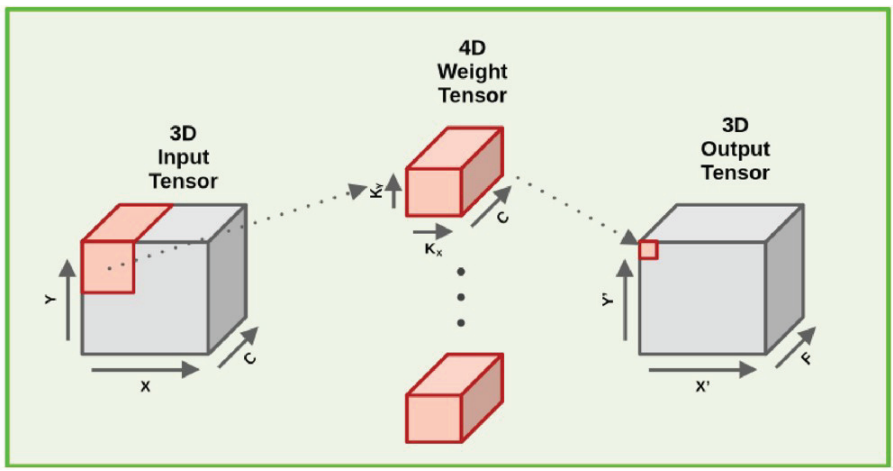

Figure 3.1 Presentation of the primary parameters of a Convolution layer in a neural network. The network consists of an input tensor with dimensions x, y and c - the channels. The convolution operation itself can be described as a tensor of size k_x and k_y with a depth of c. With F convolutions or filters, the output size can be described as x', y' and F, with x' and y' depending on stride S_x and S_y and the padding p_x and p_y.

To build the latency prediction model, we must take the following steps:

Firstly, we need to create a way to profile a neural network on the layer level. Details on how to do this will be described in 3.3.1 *Profiling*.

Using this approach, we can perform some initial representative measurements to create a generalized model of the latency behaviour of a layer with regard to single parameter changes. For this, we search for repeating patterns in the measurements and determine a function describing those. The variables of this function can be divided into global variables, which should not change for a hardware class, and local variables that depend on all other parameters of the layer. A detailed description on how to do this is given in 3.3.2 *Generalized layer model.*

From this generalized model, we can then characterize, i.e. derive a hardware-specific model for a single dimension of a neural network layer such as the number of channels in a Conv2D layer by instantiating the global and local variables of the layer. How to do this, will be described in 3.3.3 *Single Dimensional Model.*

Using the single dimension model, we expect to be able to interpolate the layer parameters one at a time. As this is a work in progress, this step has not yet been achieved.

In the following, we describe our modelling approach for GPU-based systems, as well as the resulting characterization approach.

To be able to adapt the model to different hardware types easily, we want a model that fulfils the following requirements: Firstly, it should be general, meaning it should be able to cover different hardware types without recreating the model every time. Secondly, it should require as few measurements as possible, to reduce the need for extensive characterization. Thirdly, the model should not require potentially complex or proprietary hardware knowledge for characterization.

Preliminaries The rest of the chapters use a number of terms we define here. The primary layer being discussed is the commonly used convolution 2D (conv2D) layer. The parameters relevant to latency are shown in Figure 3.1. We refer to the measurement of hardware latency used to feed the model with hardware-specific measurements as characterization. Profiling describes the measurement of a neural network and determining the single layer latencies. A (parameter) sweep is the profiling of a neural network layer, with one parameter being changed continuously while all other parameters stay constant. Finally, the model will be described as a composite function, i.e., a function whose variable is another function.

3.3.1 Profiling

For the model, we have to characterize the hardware with regards to the behaviour of single layers in the neural networks. To ensure accuracy of measurements, we measure multiple times and use the resulting confidence interval until we are sure that the mean of the samples is close enough to the mean of the distribution.

As shown in [6] we need to execute the layer in a representative network, as first and last layers absorb certain initialization and deinitialization tasks, which do not occur within a neural network. Therefore, we selected a three-layer network for profiling, with the middle layer being the layer under test.

To decrease overhead of the surrounding layers, we can perform sweeps to determine how the input and output layers affect the layer under test. If these sweeps show a convergence, the smallest layer combination, which is within a margin of the convergence point, can be selected.

Once a suitable profiling network has been determined, we perform sweeps on the parameters of the layer type to determine latency behaviour in

relation to those parameters. For Conv2D layers, the relevant parameters are input size (x, y, channels), and output filters. These sweeps are then analysed, and the model variables inferred.

This gives the model for just one layer parameter combination. To transfer this model to different layer combinations, it might be necessary to perform more measurements with other parameter combinations. Here, we perform a minimal set of profiles, under the assumption that some variables of the model, such as step size, stay constant. This can then be verified using targeted measurements at the presumed step-points.

3.3.2 Generalized layer model

To fulfil the first of our requirements, the creation of a generalized model, we first perform some sweeps along one of the parameters of a neural network layer, in this case the channel size of a Conv2D layer. We perform this with two layer configurations, varying the kernel size, to provoke a pattern change.

Based on the results shown in Figure 3.2, we can observe several trends:

Firstly, latency increases linearly with channel size. Secondly, along this linear increase the network shows a stepping behaviour with a fixed step size. Thirdly, the latency fluctuates between an upper and lower step function with a constant period.

Approximating this behaviour is possible using a nested function:

$$T(c) = \begin{cases} T_{S_U}(c), & \text{if } (c-1) \bmod r < \frac{r}{2} \\ T_{S_L}(c), & \text{else} \end{cases} \tag{3.1}$$

Here $T(c)$ is the latency of a layer based on the number of input channels c. r is the period of fluctuation. Based on this, the function selects between the step functions $T_{S_U}(c)$ and $T_{S_L}(c)$:

$$T_{S_U}(c) = \left\lfloor \frac{c}{d} \right\rfloor * d * m_u + b_u \tag{3.2}$$

$$T_{S_L}(c) = \left\lceil \frac{c}{d} \right\rceil * d * m_l + b_l \tag{3.3}$$

In these functions m_u, b_u, m_l and b_l define the upper and lower bounds for the step functions, which has a depth of d.

From Figure 3.2, we can see that the variables r and d do not change with changing layer configurations, while the upper and lower bounds are locally

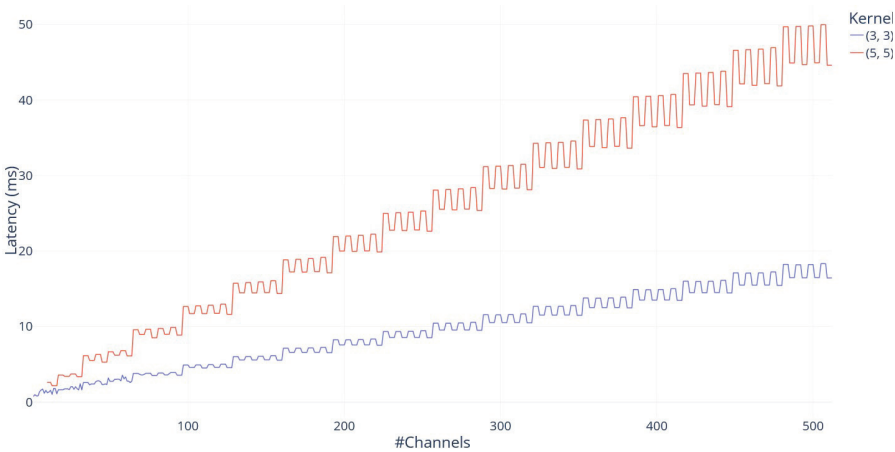

Figure 3.2 Profiling Results of Input Channel Sweep.

determined as expected, since layers with more operations should take longer than smaller layers.

For this reason, we define r and d as global variables, determined once from a sweep, and the rest as local variables, which need to be determined for each layer configuration.

3.3.3 Single Dimensional Model

To determine the values for the variables of the model for a layer dimension, we need to follow two steps:

First, we must determine the global variables. This is based on a parameter sweep as performed in the previous section. In a manual step, we can see that, e.g., for the channel sweep in Figure 3.2, the step depth appears to be 32 with a fluctuation period of 8.

In the second step, we need to determine the upper and lower bounds. This can be done by selecting values at the start and end of the sweep, lying on the upper and lower step functions as determined by the first step. Linear interpolation between each upper and lower value results in the full definition of the model for this sweep.

If this is done for each dimension of the layer, we, in theory, should be able to interpolate any layer combination using just a few simply obtained values, satisfying requirements 2 (few characterization points) and 3 (little hardware knowledge). This will have to be validated in future work.

3.4 Results

To test the approach described in Section 3.3, we did some initial characterizations of the NVIDIA Jetson AGX Orin platform. These currently cover the parameters as separate dimensions.

3.4.1 Experimental Setup

The test-setup is comprised of a NVIDIA Jetson AGX Orin running the Jetson Linux in version 36.4.4 with CUDA version 12.6. For execution and profiling we use TensorRT in version 10.3, which is a highly performant first party neural network runtime for CUDA based devices. It was chosen, because it has shown to behave predictably, changing latency as modelled.

Profiling a datapoint consists of several steps. First, we use PyTorch to create the profiling network with the required parameters and export the resulting network to the commonly used ONNX interchange format. Afterwards, this is read and converted in TensorRT to its internal format for execution on the CUDA cores of the Jetson platform. After a warmup period, we run the neural network with the profiler enabled until a sufficient certainty is obtained.

3.4.2 Characterization

For our characterization, we started with the primary dimensions (input-size, output, filters) separately.

The sweep configurations used for our initial characterization can be seen in Table 3.1. For the channel sweep the range was chosen to extend to very large parameters, to investigate if the behaviour changes at higher parameters.

In the characterization results (see Figures 3.2 to 3.4) a predictable pattern can be identified. Starting with the sweep over the channels in Figure 3.2, we can see an overall linear increase in latency w.r.t. input channels. On a lower level, we see a stepping behaviour with a step size of 32, which however changes between an upper and lower step function with a stride of 4 samples each.

Table 3.1 The sweep configurations used for the initial testing of the model hypothesis

	Input WxH	**Channels**	**Filters**	**Kernel XxY**
Channel Sweep	256x256	10 - 530	128	3x3
Filter Sweep	256x256	128	1-530	3x3
Input Sweep	10-511	64	128	1x1, 3x3, 5x5

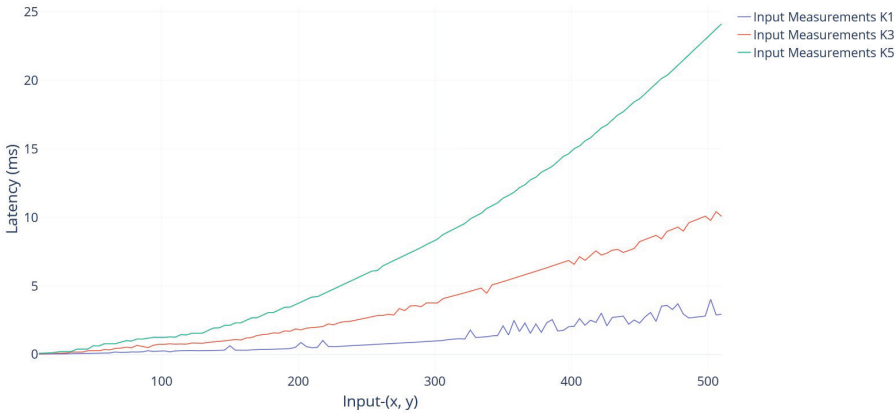

Figure 3.3 Profiling Results of Input (x, y) Sweep

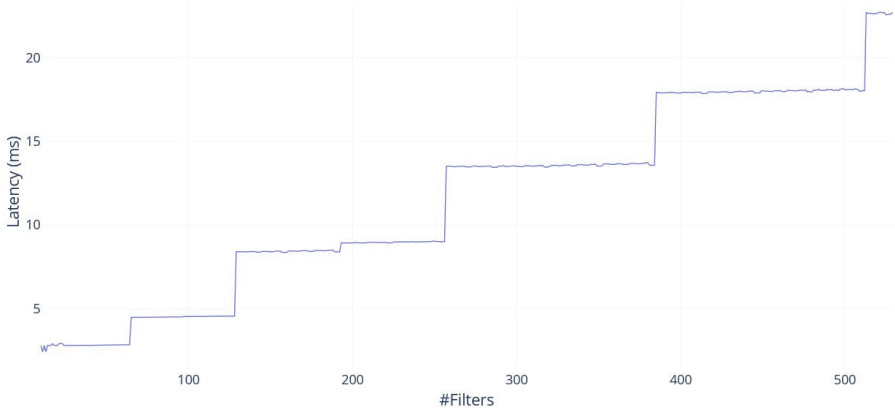

Figure 3.4 Profiling Results of Filter Sweep.

The input sweep in Figure 3.3 shows a quadratic increase over input size in each dimension. For each input dimension this results in a strictly (within the margin of error) linear increase.

For the filter sweep in Figure 3.4, again, an overall linear increase in latency can be observed with larger step sizes of mostly 128. However, in this dimension no fluctuating behaviour can be observed.

3.4.3 Initial Model Results

Our model was initially only tested for the input channel dimension.

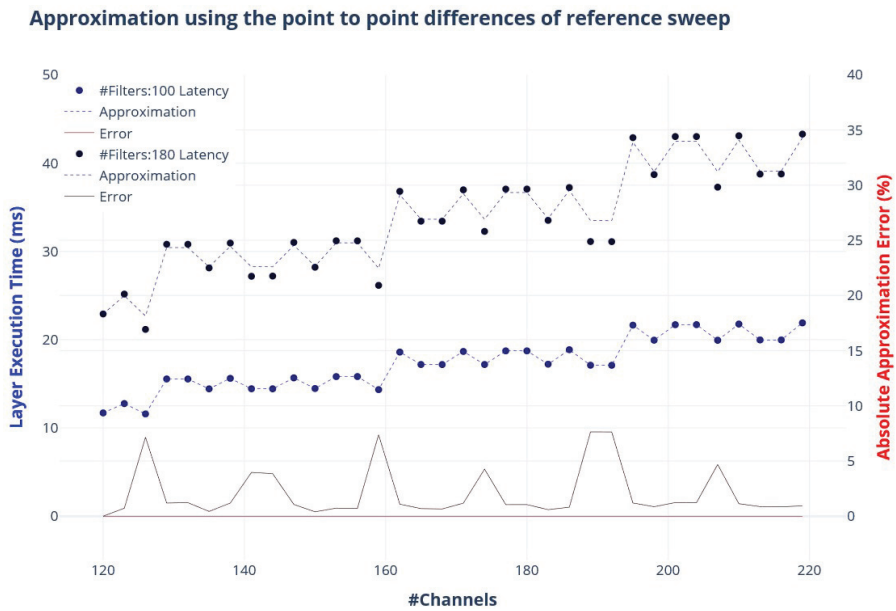

Figure 3.5 Model Results for Channels.

For testing, we used the initial channel sweep from Figure 3.2 and collected a set of sparse channel sweeps, that we tried to approximate using knowledge from the full sweep in Figure 3.2 (such as step size) and only 4 measurements, defining the upper and lower boundaries of the function.

The channel sweeps ranged from 120 to 219 with a sampling step size of 3. This sweep was performed repeatedly, only varying the output filter from 100 to 244 in steps of 16, resulting in 9 channel sweeps.

We were able to approximate the initial channel sweep with a mean absolute percentage error (MAPE) of ca. 1.6%.

The results for the smaller sweeps can be seen in Figure 3.3. In this constellation, we see an overall MAPE of ca. 1.3%.

Our evaluation shows that our method works for a single dimension, providing very accurate predictions using just the sweep data and a small number of measurements per layer configuration.

3.5 Conclusion

In this paper, we have shown that GPU-based embedded systems can, in theory, be modelled using a relatively simple hardware behaviour model and

few hardware measurements. Initial tests have shown prediction errors below 2% for single dimensions based on only a generalized characterization of the dimension and 4 additional measurements for the layer configuration.

In future work, we will extend this model to more dimensions, to be able to predict an arbitrary neural network layer configuration.

Acknowledgement

This work has received funding from the Chips Joint Undertaking (Chips JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101139892.

References

[1] D. Balemans, T. Huybrechts, J. Steckel, and S. Mercelis, "Resource-Aware Neural Network Pruning Using Graph-based Reinforcement Learning." [Online]. Available: https://arxiv.org/abs/2509.10526

[2] Q. Dariol, „Early Performance and Energy Prediction of Neural Networks Deployed on Multi-Core Platforms", Aug. 2023. doi: 10.5281/zenodo.11208197.

[3] S. Yao et al., "FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices," in Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, in SenSys '18. Shenzhen, China: Association for Computing Machinery, 2018, pp. 278–291. doi: 10.1145/3274783.3274840.

[4] E. Sotiriou-Xanthopoulos, G. S. P. Delicia, P. Figuli, K. Siozios, G. Economakos, and J. Becker, "A power estimation technique for cycle-accurate higher-abstraction SystemC-based CPU models," in 2015 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2015, Greece, Institute of Electrical and Electronics Engineers (IEEE), 2015, pp. 70–77. doi: 10.1109/SAMOS.2015.7363661.

[5] R. Raj et al., "SCALE-Sim v3: A modular cycle-accurate systolic accelerator simulator for end-to-end system analysis." [Online]. Available: https://arxiv.org/abs/2504.15377

[6] A. Osterwind, J. Droste-Rehling, M.-R. Vemparala, and D. Helms, "Hardware Execution Time Prediction for Neural Network Layers," in Machine Learning and Principles and Practice of Knowledge Discovery

in Databases, I. Koprinska, Ed., Cham: Springer Nature Switzerland, 2023, pp. 582–593.

[7] M. Wess, M. Ivanov, C. Unger, A. Nookala, A. Wendt, and A. Jantsch, "ANNETTE: Accurate Neural Network Execution Time Estimation With Stacked Models," Institute of Electrical, Electronics Engineers (IEEE), 2021, pp. 3545–3556. doi: 10.1109/access.2020.3047259.

[8] A. L.-F. Jung, J. Steinmetz, J. Gietz, K. Lübeck, and O. Bringmann, "It's All About PR," in Embedded Computer Systems: Architectures, Modeling, and Simulation, L. Carro, F. Regazzoni, and C. Pilato, Eds., Cham: Springer Nature Switzerland, 2025, pp. 59–75