Analysis of Neural Network Inference Response Times on Embedded Platforms

1st Patrick Huber
University of Applied Sciences Kempten & Technical University of Munich
Kempten, Germany
patrick.huber@hs-kempten.de

2nd Ulrich Göhner
University of Applied Sciences Kempten
Kempten, Germany
ulrich.goehner@hs-kempten.de

3rd Mario Trapp
Fraunhofer Institute for Cognitive Systems & Technical University of Munich
Munich, Germany
mario.trapp@tum.de

4th Jonathan Zender
University of Applied Sciences Kempten
Kempten, Germany
jonathan.zender@hs-kempten.de

5th Rabea Lichtenberg
University of Applied Sciences Kempten
Kempten, Germany
rabea.lichtenberg@stud.hs-kempten.de

***Abstract*—The response time of Artificial Neural Network (ANN)-inference is of utmost importance in embedded applications, particularly continual stream-processing. Predictive maintenance applications require timely predictions of state changes. This study serves to enable the reader to estimate the response time of a given model based on the underlying platform, and emphasises the relevance of benchmarking generic ANN applications on edge devices. We analyse the influence of net parameters, activation functions as well as single- and multi-threading on execution times. Potential side effects such as tact rate variances or other hardware-related influences are being outlined and accounted for. The results underline the complexity of task-partitioning and scheduling strategies while emphasising the necessity of precise concertation of the parameters to achieve optimal performance on any platform. This study shows that cutting-edge frameworks don't necessarily perform the required concertations automatically for all configurations, which may negatively impact performance.**

***Index Terms*—ANN Inference, Tensorflow Lite, Embedded Systems, Benchmarking, Response Times**

I. Introduction

The age of transformation towards industry 4.0 and transition into a data-driven society raises the relevancy of pre-filtering and intelligent evaluation of accumulated data right at the source immensely. As such, new applications for integration on edge devices become necessary. One challenge therein consists of the limited computational power (due to cost- and energy-efficiency) of edge devices [1]. Nonetheless, these devices are expected to execute intelligent and complex data evaluation. Large Artificial Neural Network (ANN)-providers satisfy these requirements by offering conversion of resource-intensive ANN models to optimised ones. One example of this is the conversion of tensorflow to tensorflow lite models. This provided the basis for benchmarking papers evaluating these models on different edge devices such as the Jetson Nano, Raspberry PI or even Smartphones. However, these papers primarily evaluate the response times of pre-trained image recognition nets like the Residual Neural Network with 50 Layers (ResNet50), Visual Geometry Group from Oxford CNN (VGG16) or the MobileNet V2 [2] on the given edge devices [3]. Their focus is on the frames per second representing the desired performance in image processing. Outside of that field, the focus is broadened (in signal processing), targeting response times of the processes. For example, in predictive maintenance, securing continual processing to predict changes in the system states is of particular interest. Overloading the system is to be avoided, while preemptive reactions should remain functional. This poses the question of which influencing factors can be used to determine the response times of processes utilising neural networks. The goal of this paper is to present empirical results for response times on given edge devices and identify their influencing factors. Therefore, we use generic and application-independent networks that enable the reader to tune networks and hardware according to their own specific use case and requirements. Thus, we address the lack of comprehensive benchmarking studies focusing on the response times of generic ANN models on edge devices. We differentiate our work from real-time capability as the focus is on timely, but not necessarily immediate, predictions. We provide value by evaluating the capability of these systems to continually process signals without delay or bottlenecks. This facilitates a timely yet potent prediction of state changes, which can contribute to the improvement of predictive maintenance.

This paper is structured into five sections following the introduction, the first of which revises net dimensions & problem complexity under section II to categorise the chosen dimensions of the evaluated networks. The experimentation setup is presented in section III, followed by an analysis of the influence of network structure on the execution times in section IV. That analysis allowed for the reduction of experimentation parameters based on relevance. The experiments are conducted in section V, followed by a conclusion in section VI.

II. Net Dimensions

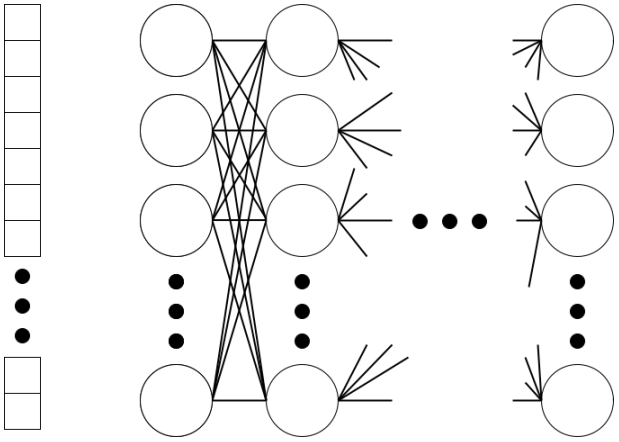

Fig. 1: Rectangular Neural Network Architecture with different Input Dimensions

A. Review of Net Dimensions & Problem Complexity

Classic image processing often utilises highly complex networks containing hundreds of thousands, if not millions of weights in order to solve such complex problems. Sizing of the net dimensions for the experiments in this work is tailored to signal processing (e.g. predictive maintenance), however. Time series forecasting or anomaly detection as performed here usually don't require millions of weights [4] [5] [6].

For example, the paper [7] trains a Multi Layer Perceptron (MLP) for predictive maintenance in substations, identifying eleven influencing factors that were evaluated using neural networks. These networks consisted of eleven input neurons, at most 20 layers and one output neuron. The paper [4] forecasts the highest temperature to be expected in South Korea, evaluating multiple networks with between 49 and 1001 neurons for that prediction. Meanwhile, the paper [5] predicts the Key Performance Indicators (KPI) of computers using a very simple neural network with five layers.

These examples demonstrate the relevance of such networks in real world applications. Research proves that many networks under 40,000 neurons provide a balanced accuracy with performances within the range of 98% to 99.5% [6].

B. Sizing of the Experiments' Net Dimensions

The networks chosen for the experiments are all equivalent in their shape. All networks evaluated here are rectangular and fully connected. Their general architecture is pictured in Figure 1. In sizing the net dimensions for our experiments, we settled on a compromise between the aforementioned dimensions for image and signal processing from Subsection II-A. This serves to ensure representation of the majority of signal processing applications by over-sizing.

The influencing factors of net dimensions encompass the amount of layers, neurons per layer as well as input and output dimensions, which vary in strides of ten each as follows:

- Input/output dimensions between 1 and 91 (10 variations)
- Neurons per layer between 2 and 192 (20 variations)
- Layers between 2 and 192 (20 variations)

As a result, there are 4,000 different network configurations, of which the most complex counts over seven million trainable weights, while the simplest has seven. The term layer herein covers input, output and hidden layers. Subsequently, a network with two layers merely contains one input, one output and no hidden layers.

III. Experimentation Setup

A. Time Measurement

Walltime is used to determine response times, which is justified as it allows for equivalent response time measurement for single and multi-threading. In order to minimise potential distortion trough interrupts, we perform 100,000 calculations (invokes) with varying input vectors and calculate the average duration of those. This approach is inspired by the works of [8] and [2], who rely on Walltime as well. Aside from executing the networks on one Central Processing Unit (CPU) core of the edge device, we also measured response times on multiple cores. However, we did not implement the multi-threading ourselves, but initialised the tensorflow lite inference with multiple threads enabled.

B. Data Set

The subsequent experiments utilise synthetically generated data as our focus lies on measuring response times, not evaluating the quality of the classifications. A causal reduction of weights by e.g. pruning is not considered because the standard converter implementation from tensorflow to tensorflow lite does not account for such optimisations [9]. As such, all neurons remain in the model and generate calculation effort. Subsequently, the goal is to evaluate the performance of the hardware using the tensorflow lite models rather than the performance of potentially system-specific delegators or quantification methods.

C. Evaluation Hardware

Benchmarking was performed on the systems listed in Table I. All experiments were run on the internal CPU of the system. Hardware acceleration was omitted due to focusing on small networks, as its initialisation creates an overhead and produces additional costs for read/write operations on memory [10]. In addition, hardware accelerators such as the Neural Processing Unit (NPU) of the 8MPLUSLPD4-EVK require quantification of the models since unsupported operations cause hopping between the CPU and accelerator, negatively impacting response times [10]. As the application of such quantifications effects the precision of the models, it stands in opposition to the goals set in Subsection III-B.

Name	NXP 8MPLUSLPD4-EVK	Raspberry Pi 4 Model B	NVIDIA Jetson AGX XAVIER
Processor	ARM Cortex-A53 4 Core	ARM Cortex-A72 4 Core	NVIDIA Carmel ARM 8 Core
Clock Speed	1.80 GHz	1.50 GHz	2.20 GHz
Operating System	Yocto 5.15 (kirkstone)	Debian 11 (bullseye)	Ubuntu 20.04.6 (focal)

TABLE I: Benchmarking Hardware

IV. Analysis of the Influence of Net Dimensions and Structure

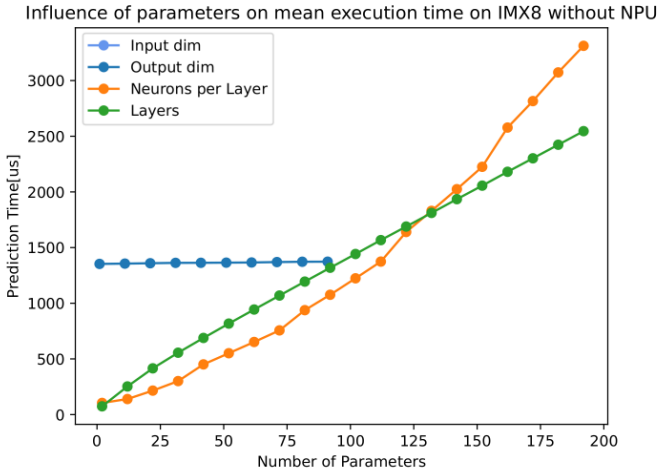

Fig. 2: Average response times on IMX8 grouped by input dimension, neurons and layers

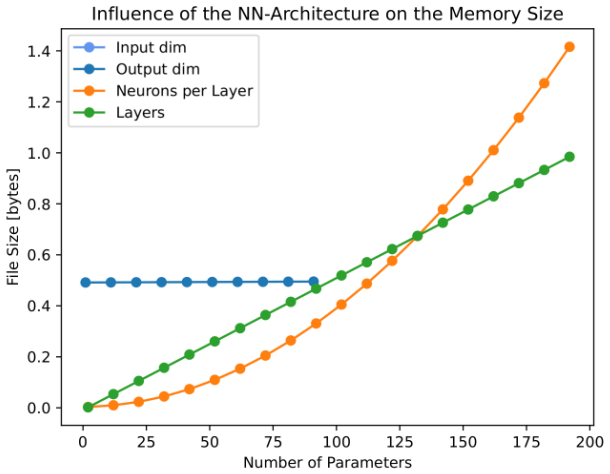

Fig. 3: Memory usage grouped by input dimension, neurons and layers

This section inspects the influencing factors of the net dimension in accordance with II-B for response time and memory usage. Figure 2 shows the impact of each factor on average response times, sorted by neurons per layer, layers and input/output dimensions. It is noteworthy that input/output dimensions only marginally influence the average response time over all variations. As the input and output dimensions overlap in this representation, merely the output dimension is evident. Additionally, we could prove through the experiments that the memory usage of the influencing factors has an equivalent impact on response times, as shown in Figure 3.

The correlation between response times and memory usage implies dependence on the total amount of trainable parameters. Accordingly, (1) - (4) are introduced to classify the results: Calculation of trainable parameters for the input layer p_{il} (see (1)) as well as the output layer p_{ol} (see (2)) is performed utilising the corresponding dimension (dim_i and dim_o) and the amount of neurons n. Furthermore, the parameters of the hidden layers p_{hl} are determined under consideration of the total amount of layers l (see (3)). Subsequently, the sum of all parameters p_t is calculated as seen in (4).

$$p_{il} = (dim_i + 1) * n \tag{1}$$

$$p_{ol} = (n + 1) * dim_o \tag{2}$$

$$p_{hl} = [(n + 1) * n] * (l - 2) \tag{3}$$

$$p_t = p_{il} + p_{hl} + p_{ol} \tag{4}$$

For Equations (1) - (3), the bias of the previous layer is taken into account through incrementation. Subsequently, in accordance with (3), there is a quadratic approximation of neurons per layer to the trainable parameters as well as a linear incline of layers, which aligns with the measurements found in Figure 2. We found a linear correlation between the average response times and total trainable parameters per network, indicating a strong dependence of these response times on the total parameters. Therefore, the empirical results (see Section V-B) are presented in the form of comparison between the two.

V. Conducting the Experiments

A. Experimentation Method

The empirical results are generated for the networks introduced in Section II-B on the chosen hardware platforms (see Section III-C). In this section, the applied experimentation method complements the framework conditions of the existing experimentation setup outlined in Section III, based on the previous findings concerning the influence of net dimensions and structure from Section IV. Accordingly, the framework conditions for the evaluation are defined as follows:

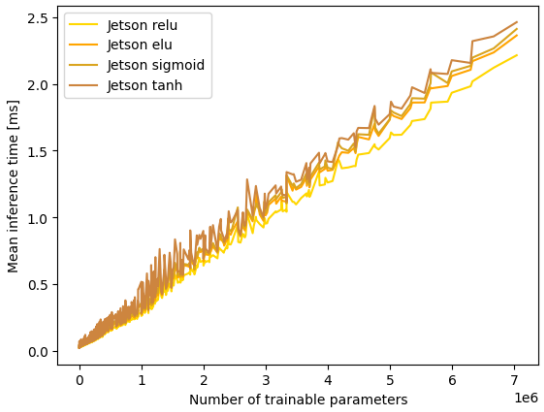

Fig. 4: Comparison between activation functions on Jetson single core

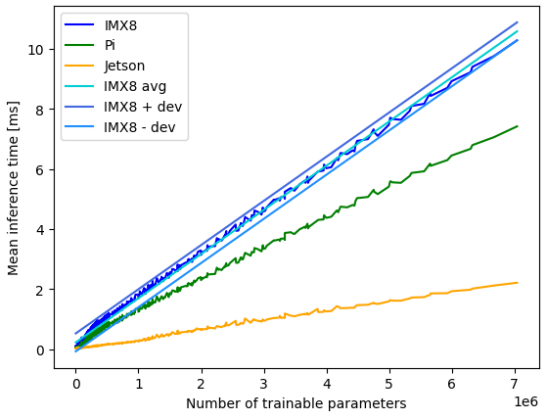

Fig. 5: Single-thread ReLU performance across all architectures including averages and deviations

- As response times strongly depend on the total amount of trainable parameters in a network as well as the hardware platform, these dimensions are to be compared.
- In order to reduce the amount of variations, networks are structured as rectangles because the amount of trainable parameters, as well as the response times, of e.g. pyramidal networks are enclosed in those of rectangular nets.
- For further reduction of experiments, input and output layer variations were omitted and set to a constant of one due to the marginal influence on response times.
- Since a network consisting of twelve layers and two neurons per layer has the same amount of trainable parameters (67 total) as one consisting of two layers and 22 neurons, networks with equivalent amounts of total trainable parameters are not measured anew.
- No processes aside from those necessary to the operating system were run concurrently with the benchmarking in order to minimise the impact of outliers (e.g. interrupts), as well as running 100,000 calculations each.

Initial experiments under these conditions proved the impact of varying activation functions and subsequent changes in calculation operations on response times, as expected. This behaviour is represented in Figure 4, displaying the single-thread performances on the Jetson architecture. Accordingly, one experiment was conducted for the most common activation functions Rectified Linear Unit (ReLU), Exponential Linear Unit (ELU), Sigmoid and Tangens Hyperbolicus (TanH) each. Due to the linear correlation described in Section IV, and because displaying all measurement results would be unhelpful due to the extensive scope, our results will be approximated via linearisation. Additionally, we provide the maximum deviation in ms in relation to the aforementioned linearisation. This procedure permits approximations for the given hardware platforms and activation functions outside of the chosen net dimensions (number of trainable parameters) by linear extrapolation. The precision of measured data is nanoseconds, which is why the results are given in milliseconds with six decimal places.

Hardware Platform	Activation Function	Gradient	Y-Axis Section	Maximum Deviation
IMX8	ReLU	1.469174e-6	2.303959e-1	0.298859
	ELU	1.520086e-6	3.118073e-1	0.365074
	Sigmoid	1.524141e-6	3.135349e-1	0.378036
	Tanh	1.649902e-6	4.050808e-1	0.553396
Raspberry Pi	ReLU	1.071348e-6	1.578266e-1	0.286284
	ELU	1.112801e-6	1.998119e-1	0.345382
	Sigmoid	1.114504e-6	2.092044e-1	0.326310
	Tanh	1.151102e-6	2.668625e-1	0.392679
Jetson	ReLU	3.188076e-7	1.063447e-2	0.153478
	ELU	3.409927e-7	2.067891e-2	0.192275
	Sigmoid	3.475815e-7	2.299473e-2	0.164944
	Tanh	3.579973e-7	4.256916e-2	0.274846

TABLE II: Gradients, y-axis sections and maximum deviation coefficients for all activation functions across all architectures in single-threading

B. Empirical Results

1) Single-Threading: Figure 5 illustrates the classification of response times across the different hardware platforms. For improved readability, only the ReLU measurements are given. Additionally, this figure shows the linearisation for the IMX8 via avg. Since the relation of response times to total trainable parameters has a linear trend, yet does not incline monotonously, we consider $-dev$ as well as $+dev$ to emphasise this fact. In addition, the corresponding absolute value of the maximum deviation in relation to the linearisation is provided.

Analysis of the empirical results as shown in Table II proves minimal values for both the gradient and y-axis sections in case of the ReLU activation function on the Jetson architecture, leading to minimal response times and according best performance. Values for the gradient and y-axis sections imply ascending order of activation functions in regard to response times across all hardware platforms as follows:

1) ReLU
2) ELU
3) Sigmoid
4) TanH

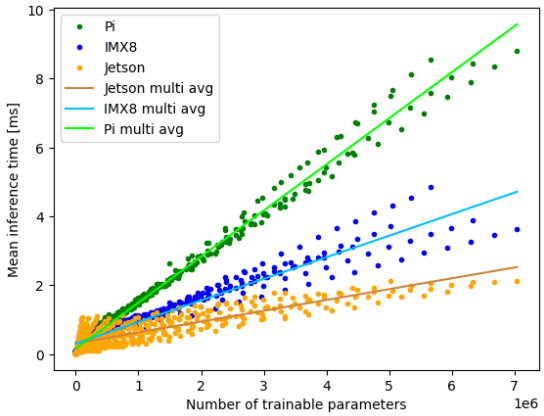

Fig. 6: Multi-thread ReLU performance across all architectures including averages

Hardware Platform	Activation Function	Gradient	Y-Axis Section	Maximum Deviation
IMX8	ReLU	6.260788e-7	3.092691e-1	1.075668
	ELU	6.764540e-7	3.874742e-1	1.168902
	Sigmoid	6.788531e-7	3.905559e-1	1.129849
	Tanh	9.248984e-7	1.003794e+0	1.746782
Raspberry Pi	ReLU	1.337886e-6	1.586471e-1	0.815641
	ELU	1.415562e-6	2.039669e-1	1.476009
	Sigmoid	1.399550e-6	2.106196e-1	1.007297
	Tanh	1.450218e-6	9.194122e-1	2.250828
Jetson	ReLU	3.143797e-7	3.164306e-1	0.715131
	ELU	3.473014e-7	3.510930e-1	0.769705
	Sigmoid	3.550585e-7	3.552569e-1	0.814380
	Tanh	6.556785e-7	1.190182e+0	2.846824

TABLE III: Gradients, y-axis sections and maximum deviation coefficients for all activation functions across all architectures in multi-threading.

Additionally, the hardware platforms can be sorted in regards to response times for single-threading in ascending order as shown in Table IV. The maximum deviations vary between around 0.15ms and 0.55ms. Values for the maximum deviation in Table II imply a reduction of deviation in case of improved hardware performance, however, we deem the impact of activation functions on the deviations too insignificant to draw conclusions.

2) Multi-Threading: The experiments were repeated for multi-threading, using four threads each for the sake of comparability. Figure 6 illustrates the classification of response times for the ReLU activation function across different hardware platforms. For the comparison between multi- and single-threading (see Section V-C), we provide the linearisation via avg and maximum deviation like before.

It is noteworthy that the platforms IMX8 and Raspberry Pi generate linear groups of measurement results. Despite the fact that both architectures possess four CPUs each, the Raspberry Pi only generates three such prevalent lines, while the amount of lines is in accord with the amount of CPUs on the IMX8. This behaviour indicates a possible difference in task scheduling; potential influencing factors for this are

Rank	Single-Threading	Multi-Threading
1	Jetson	Jetson
2	Raspberry Pi	IMX8
3	IMX8	Raspberry Pi

TABLE IV: Ranking of hardware platforms for single- and multi-threading based on response times in ascending order

presented under Section V-D. Analysis of the empirical results as shown in Table III based on the gradient and y-axis sections prove minimal response times and according best performance for the ReLU activation function on the Jetson architecture.

The impact of the activation functions on response times generates differentiated behaviour in case of ELU and Sigmoid on the Raspberry Pi. While a low total of trainable parameters lead to a higher response time for the Sigmoid activation function compared to ELU (see y-axis sections), this behaviour inverts upon increasing the total amount of trainable parameters, due to the low gradient inclination. Aside from the aforementioned differential behaviour, the measured values imply equivalent sorting of activation functions in regards to response times as seen in single-threading (compare Subsection V-B1).

Additionally, the hardware platforms can be sorted in regards to the response times according to Table IV for multi-threading. The maximum deviations vary between around 0.72ms and 2.85ms. Analysis of the deviations shows no significant correlation to platform performance, meaning that the scattering does not necessarily align with the platforms potency. Deviations are minimal for the ReLU activation function on all platforms, while the TanH always generates the greatest deviations. Maximum deviations correlate with net dimensions for the IMX8 and Raspberry Pi, meaning greater scattering for greater networks. Due to the aforementioned forming of lines, we propose the hypothesis of different task partitioning and scheduling strategies as the cause. For this reason, we will take a closer look at this line formation in Section V-D.

C. Comparison between Multi- and Single-Threading

Empirical results show a differentiated behaviour in regards to the deviation when comparing multi- and single-threading. There exists a general increase in scattering for multi-threading when compared to single-threading. Additionally, scattering is also influenced by the hardware platform, dependant on the total amount of trainable parameters. This varying behaviour becomes evident in the comparison of deviations on the IMX8 and Jetson, as shown in Figure 6. In order to maintain clarity for the multitude of variations, we retained the measure for the deviation previously introduced under Subsection V-A. Therefore, when using linearisation as extrapolation, the deviation values are undetermined for multi-threading in contrast to single-threading.

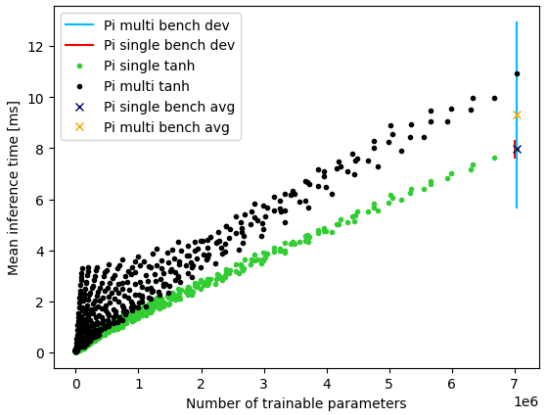

Fig. 7: Raspberry Pi TanH performance on single- and multi-thread compared to average and deviation obtained from library benchmarking tool using largest model

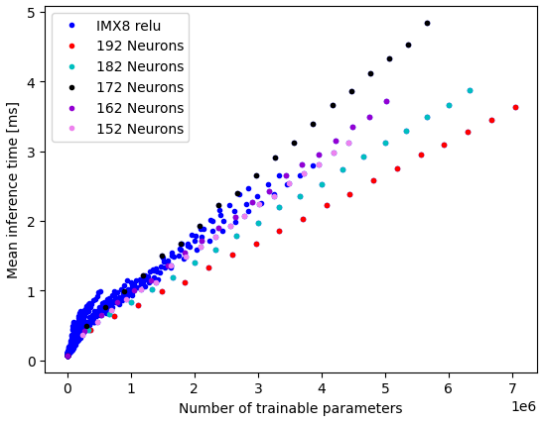

Fig. 8: IMX8 performance on multiple threads with selected individual lines based on neurons

Different activation functions generate differentiated scattering as well, despite running on the same hardware. For example, the maximum deviation varies between 0.72ms (ReLU) and 2.85ms (TanH) for multi-threading on the Jetson, while single-threading varies far less, between 0.15ms (ReLU) and 0.27ms (TanH). However, response times have shown unexpected behaviour. Only the IMX8 architecture wholly reduces response times through multi-threading as expected. Meanwhile, against the expectations, the Jetson generated a noticeable increase in gradient by about 183.24% for the TanH while also increasing y-axis section value, resulting in delayed response times. The variations in response times for the remaining activation functions on the Jetson are lesser, yet imply ineffective task partitioning and scheduling for multi-threading when compared to single-threading.

The Raspberry Pi consistently produced higher response times across all activation functions when using multi-threading. In addition to the worsening of response times, the reduction of the latter on the IMX8 causes it to overtake the Raspberry Pi for multi-threading, as seen in Table IV. Taking the CPU Mark for the integrated processors into account, wherein the ARM Cortex-A72 4 Core (Raspberry Pi) outperformed the ARM Cortex-A53 4 Core (IMX8) [11], the user would not expect such behaviour. In order to eliminate the possibility of systematic errors on our end, we chose to compare our data with an alternative benchmarking tool, for which the onboard tensorflow lite benchmark tool was utilised. This tool is tailored to producing empirical results as exact as possible for any given model. To that end, for example, specific warm-up invokes are performed in advance [12]. However, it is not suited to measuring many varying models, as was our use case. Figure 7 shows the comparison of behaviours for the Raspberry Pi using the TanH activation function. Additionally, dev and avg of the tensorflow lite benchmark tool (referred to as $bench$) were included, measured for the largest model. Ultimately, the tool supports our findings as its measurements generate a significant increase in response times in case of multi-threading compared to single-thread execution as well. As was the case for our measurements, the tool's measured deviation increased for multiple cores compared to running on just one.

D. Side Effects

In the previous sections, it was proven that both the hardware platform as well as the activation function cause differentiated behaviour in regards to response times. Additionally, the choice of threading partially had an unexpected impact (see e.g. Raspberry Pi in Subsection V-B2). Subsequently, further potential side effects were analysed during the experiments. Logging of the tact rate eliminated the possibility of minimum and maximum tact rate span causing variations in execution times. Furthermore, we had the hypothesis that simultaneous use of multiple cores could result in changes in system temperature. This is potentially supported by the different cooling systems. When comparing the tact rates of the ARM Cortex-A72 4 Core (Raspberry Pi) and ARM Cortex-A53 4 Core (IMX8) as seen in Table V, at first, one might think it implies an explanation for the worsening of response times, but our logs showed no sign of tact rate throttling during the experiments. Furthermore, the logs didn't reveal any RAM bottlenecks that could lead to increased page faults.

Subsequently, the illustrated increase in execution times in multi-threading implies partially inefficient task partitioning and scheduling. These could facilitate a pipeline hazard, potentially further increasing execution times in addition to the scheduling overhead. Due to the heterogeneous hardware architectures in regards to e.g. caching, individual identification of influencing factors requires detailed inspections well beyond the frame of this work.

	Cortex-A72	Cortex-A53
CPU max MHz	1500	1800
CPU min MHz	600	1200

TABLE V: Tact rates as measured by lscpu command

During our inspection of the results, we managed to select individual lines from the multi-threading plots by amount of neurons per layer (see Figure 8). This further supports our thesis concerning inefficient task partitioning and scheduling, as the network structure is one of its influencing factors. This is illustrated by the fact that there is no direct correlation between the number of neurons and the reaction time. However, there is a linear trend for any number of neurons per layer. In conclusion, it is important to remember that the optimal amount of threads depends on a multitude of factors like the means of calculation, CPU architecture, type of model and available resources.

VI. Conclusion

The present work commits to an in-depth analysis of response times of ANN inferences on embedded platforms. It mainly analysed the impact net dimensions, activation functions and the execution as single- or multi-thread inference have on response times. The empirical data permitted approximation of response times for ANN models on the chosen platforms. These approximations allow the user to configure their model for continual stream processing. As such, this paper answers an unmet demand in benchmark research by extending focus from existing image processing networks to generic ones that can be used in diverse application domains. Additional findings of our study underline the complexity of ANN model optimisation for edge devices. We researched the impact of net dimensions, including layers, neurons per layer as well as input and output dimensions, on response times. Our experiments consisted of a varied series of net configurations which enabled us to identify patterns in the relationship between net structure and execution times. The choice of activation functions played into determining the response times as well. We compared the ReLU, ELU, Sigmoid and TanH activation functions and showed the influence they have on response times on different hardware platforms.

In addition, we researched the impact of execution as single- versus multi-thread on different hardware platforms. Analysis proved the dependence of choice of optimal execution mode on multiple factors. Contrary to our expectations, in some cases multi-threading lead to significantly higher response times. This uncovers a need for optimisation in regards to partitioning and scheduling strategies in state-of-the-art libraries. In conclusion, this study adds to the understanding of the challenges that are tied to the utilisation of ANN models on embedded platforms with limited resources. Since demand for ANN applications at the source of data keeps increasing, our findings may facilitate making well-founded decisions for reaching optimal performance in various real scenarios.

References

[1] R. Khandelwal, "A Basic Introduction to TensorFlow Lite - Towards Data Science," USSR, Dec. 2021. [Online]. Available: https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292

[2] S. P. Baller, A. Jindal, M. Chadha, and M. Gerndt, "Deepedgebench: Benchmarking deep neural networks on edge devices," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 20–30.

[3] C. Luo, X. He, J. Zhan, L. Wang, W. Gao, and J. Dai, "Comparison and benchmarking of ai models and frameworks on mobile devices," *arXiv preprint arXiv:2005.05085*, 2020.

[4] T. T. K. Tran, T. Lee, and J.-S. Kim, "Increasing neurons or deepening layers in forecasting maximum temperature time series?" *Atmosphere*, vol. 11, no. 10, p. 1072, 2020.

[5] A. Acker, T. Wittkopp, S. Nedelkoski, J. Bogatinovski, and O. Kao, "Superiority of simplicity: A lightweight model for network device workload prediction," in *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2020, pp. 7–10.

[6] V. Krasteva, S. Ménétré, J.-P. Didon, and I. Jekova, "Fully convolutional deep neural networks with optimized hyperparameters for detection of shockable and non-shockable rhythms," *Sensors*, vol. 20, no. 10, p. 2875, 2020.

[7] I. Ullah, F. Yang, R. Khan, L. Liu, H. Yang, B. Gao, and K. Sun, "Predictive maintenance of power substation equipment by infrared thermography using a machine-learning approach," *Energies*, vol. 10, no. 12, p. 1987, 2017.

[8] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–10.

[9] TensorFlow, "Modelloptimierung | TensorFlow Lite," May 2021, [accessed 25. Oct. 2022]. [Online]. Available: www.tensorflow.org/lite/performance/model_optimization

[10] G. Verma, Y. Gupta, A. M. Malik, and B. Chapman, "Performance evaluation of deep learning compilers for edge inference," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 858–865.

[11] PassMark Software Inc., "ARM Cortex-A53 4 Core 1800 MHz vs ARM Cortex-A72 4 Core 1500 MHz [cpubenchmark.net] by PassMark Software," Oct. 2022, [Online; accessed 26. Oct. 2022]. [Online]. Available: https://www.cpubenchmark.net/compare/4128vs3917/ARM-Cortex-A53-4-Core-1800-MHz-vs-ARM-Cortex-A72-4-Core-1500-MHz

[12] TensorFlow, "Performance measurement," Sep. 2022, [Online; accessed 6. Dec. 2023]. [Online]. Available: https://www.tensorflow.org/lite/performance/measurement