Bang for the Buck: Evaluating the cost-effectiveness of Heterogeneous Edge Platforms for Neural Network Workloads

Amarjeet Saini*
Omkar B Shende *
amarjeets167@gmail.com
212011004@iitdh.ac.in
Department of Computer Science and Engineering
Indian Institute of Technology, Dharwad, Karnataka
India

Mohammad Khalid Pandit
Department of Computer Science and Engineering
Indian Institute of Technology, New Delhi
India
khalid@cse.iitd.ac.in

Rijurekha Sen
Department of Computer Science and Engineering
Indian Institute of Technology, New Delhi
India
riju@cse.iitd.ac.in

Gayathri Ananthanarayanan
Department of Computer Science and Engineering
Indian Institute of Technology, Dharwad, Karnataka
India
gayathri@iitdh.ac.in

Abstract

Machine learning (ML) applications have experienced remarkable growth and integration into various domains. However, challenges with cloud-based deployments, such as latency, privacy, reliability, bandwidth and connectivity, have driven the popularity of deploying ML on edge devices. ML application deployment stack consists of various components such as neural network models, input frameworks, software runtime libraries and hardware architecture. Understanding the impact of different components in the ML stack on deployment effectiveness, particularly in terms of cost effectiveness, remains a challenge. In this work, we systematically analyze the diverse choices available for each component of the ML stack and their influence on deployment performance. We empirically evaluate eight heterogeneous edge platforms and eight software runtime libraries, considering various hardware components like CPUs, GPUs, NPUs, and VPUs for ML inference. Our findings contribute to a better understanding of optimizing cost effectiveness in ML deployments on edge platforms, aiding decision-making for application developers and stakeholders.

CCS Concepts: • **Computer systems organization → Embedded hardware**; • **Computing methodologies → Machine learning**; *Computer vision.*

Keywords: Neural networks, edge computing, accelerators

ACM Reference Format:
Amarjeet Saini, Omkar B Shende, Mohammad Khalid Pandit, Rijurekha Sen, and Gayathri Ananthanarayanan. 2023. Bang for the Buck: Evaluating the cost-effectiveness of Heterogeneous Edge Platforms for Neural Network Workloads . In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

*Both authors contributed equally to this research.

Conference acronym 'XX, December 06–09, 2023, Wilmington, DE

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Neural Networks have revolutionized the way society functions, in recent years. While training of Neural Networks (NN) uses cloud computing, real time NN inferences predominantly happen at the "edge" i.e. inside a self driving car or on a traffic pole of a smart city intersection, closer to where data to be processed is collected. The rapid pace at which such edge intelligence is being adopted in various application domains like smart homes, automotive, healthcare etc. [1, 20], have fueled the interest in hardware vendors to design and manufacture specialized hardware to accelerate ML workloads on the edge devices [3]. In this work, we do a deep dive assessment of NN inferences on edge platforms with such state-of-the-art ML workload accelerators.

Heterogeneity in compute engine architecture is essential to cater to diverse types of workloads and thus vendors include more than one type of compute engine (namely CPU, GPU, VPU, NPU, TPU) on the SoC. This variety is termed as *accelerator heterogeneity.* Additionally, most vendors usually provide bundled software accelerated runtimes [4, 6, 14, 19] comprising of NN model converter, optimizer, a library of computation kernels and support for variety of input NN frameworks. *Which edge platform to choose for a target NN application? Do the platforms in similar price points perform the same? If the chosen platform has more than one processor or ML accelerator, where should the NN inferences be run? Are there many software frameworks available for the chosen edge platform? Is there any benefit of choosing one software framework over another? Does it make sense to involve more than one processor concurrently in NN inferencing? If yes, how should the workload be split among the concurrently running co-processors? If my application metrics change to include energy minimization or thermal safety, would my choice of platform, processor and software stack that optimized for throughput metrics earlier, become invalid?* The landscape available for designing an edge solution with NN inference, is very vast and complex. We carefully navigate this design space exploration problem using both theoretical analysis and empirical measurements in this paper.

While some measurement studies have been conducted in recent past [22, 25, 27, 30, 39, 42, 44, 47] to understand ML performance

at edge, there are significant limitations in their scale and comprehensiveness. We have used eight different hardware platforms in this study. Some of these platforms have heterogeneous processors and accelerators of similar compute capabilities, while in others the capabilities vastly differ. Some platforms have all processors co-located on the same chip, whereas in others, external accelerators can be plugged in over USB. In addition to this rich variety of hardware platforms and NN accelerators, we compare a wide range of the state-of-the-art software runtimes like *ARM NN*, *TensorRT*, *OpenVINO™*, *KSNN*, *RKNN*, *TensorFlow Lite* and *AutoScheduler*. We also use realistic NN workloads like more than 12 standard NN models from the Keras model zoo to run inferences on the ImageNet dataset. We even present a real edge NN application of vehicle detection, using a novel traffic intersection camera dataset from Delhi, India. The sheer scale of edge hardware platforms, software frameworks, NN models and datasets, allow us to do very interesting analyses, unseen in prior benchmarking efforts.

In this work, we present some insightful findings with a broader applicability across different hardware platforms ❶ `CPU Bottleneck`-High-end accelerators' performance can be limited by underpowered CPU cores as CPU handles the input/output for NN inferences on the accelerator. Our studies reveal that most off-the-shelf edge platforms often exhibit over-investment on accelerator performance, with insufficient attention given to the commensurate enhancement of CPU cores responsible for driving these accelerators. ❷ The `Energy-Delay-Cost-Product`(EDCP) metric, introduced in this study alongside accuracy metric, can serve as a robust tool for evaluating available options and aiding in the selection of one platform over another for practical application deployment.❸ It is often expected that the vendor provided software frameworks always perform better as hardware vendors possess intricate knowledge of their manufactured chips. We empirically show that open-source software frameworks often outperform vendor-provided software for specific chips and insider knowledge of hardware details can be eclipsed with better software implementation skills. ❹ Concurrent use of more than one accelerator need not boost inference throughput compared to solo run on a single processor. Resource contention during concurrency, compute speed mismatch among processors, communication delays between processors, are some root causes for performance degradation under concurrency. ❺ Data wise workload allocation works better with off-chip accelerators than with the on-chip accelerators and layer wise workload allocation suffers performance degradation due to inter-process communication. In addition, we also show through our application case study that even a slow CPU processor can be used extremely effectively with faster accelerators in a heterogeneous platform, to achieve delicate trade-offs among metrics. Our methods and results should be of interest to hardware vendors, software framework engineers, edgeML researchers and solution architects deploying NN inferences at edge for various application domains.

2 Related Work

Benchmarking machine learning workload performance in edge devices is an active field of research. The work in [48] provides a comprehensive catalog of existing research related to edge performance benchmarking. These benchmarking efforts can be broadly classified under three categories, **`(1) ML algorithm benchmarking`**, **`(2) performance benchmarking`**, **`(3) ML-system benchmarking`**. ML algorithm benchmarking focuses on algorithmic innovations and application utility metrics like accuracy [33] [28] [41] [34]. Performance benchmarks focus on hardware optimization and performance metrics like power consumption, latency, and throughput [32] [31] [21]. The ML-system benchmarks lie at an intersection, focusing on both utility and performance [22, 25, 27, 30, 39, 42, 44, 47]. Our work lies in the third category, where we examine NN inference accuracy as well as performance on edge platforms.

Two features distinguish this paper compared to prior works. First is **`the scale of the study`**. We combine theoretical roofline analysis with real empirical measurements on a wide variety of edge devices (Table 2) with general purpose processors like CPU, GPU and a wide range of NN accelerator chips from all major hardware manufacturers like Nvidia, Intel, ARM, Verisilicon and many more while prior works have compared a maximum of 2-3 platforms. Second distinctive feature is **`benchmarking workload allocation across heterogeneous co-processors`**. With the emergence of multiple processing elements on edge devices, their optimal usage and workload allocation among them, are sought after problems in the research community. Some of the existing studies [38] [46] [40] [51] investigate workload allocation, examining it from diverse angles such as inference parallelization, optimizing throughput, and enhancing resource utilization.

While many existing benchmarking efforts tend to focus primarily on one of three key aspects: SW framework analysis, workload allocation, or quantization support, it's crucial to recognize the interplay among these choices. Different software frameworks enable diverse approaches to workload allocation and quantization. In this study, we delve deep into these intricate dependencies, meticulously examining the impact of various workload allocation strategies on the execution of ML workloads across heterogeneous co-processors. Furthermore, we conduct a comprehensive evaluation encompassing a range of processor combinations, experimenting with various workload allocation strategies across both vendor-specified and open-source software frameworks that offer varying levels of quantization support for optimal inference performance. We work with all possible software frameworks on approximately 20 different types of processing elements on 8 different hardware platforms, that exceeds the scale of prior benchmarking efforts manifold. Furthermore, to provide a robust assessment, we juxtapose our findings from the roofline model with systematic empirical measurements conducted on the target hardware platforms. The detailed comparison of prior work is summarized in the Appendix in Table 8, which showcases the earlier NN system benchmarking efforts and lists the evaluation criteria used in the study.

3 Navigating the imperatives

There is a huge surge in the adoption of Edge ML based solutions in various industries like healthcare, manufacturing, retail etc [37]. However, the entire stack that makes up the Edge ML solution is very complex in nature. This makes the process of picking the right choices for deploying the solution, non-intuitive and a difficult task. Do the highest priced platforms always perform better ? What are the factors that dictate the performance of the deployed application ? In this section, we try to break down some of the most important aspects to consider when evaluating an Edge ML solution for deployment.

`(a) Choice of the NN Model:` The first decision to make is in

terms of choosing the appropriate NN model for realising the objectives. The application developer will need to carefully sift through potential trade-offs between factors like model size, accuracy and throughput of various models before converging on the NN model. *In this paper we have experimented with more than 12 standard DNN models from Keras model zoo [11]. For brevity, we present our results with four selected NN models for image classification, that have a wide range of model size, layer numbers and parameters and YOLOv3 model for object detection (Table 1) .*

NN Model	#Parameters (Millions)	Depth	Model Size (MB)	FP32	FP16	INT8
MobileNet-V2 [45]	3.5	105	14	1.33	1.63	1.93
ResNet101-V2 [35]	44.7	205	171	1.6	1.9	2.2
DenseNet-121 [36]	8.1	242	33	1.94	2.24	2.54
Xception [29]	22.9	81	88	1.96	2.26	2.56
Yolov3 [43]	62.2	106	246.6	2.42	2.72	3.02

Table 1. NN models for vision based tasks. Operational Intensity (FLOPS/Byte) for different precisions. Model size denoted here is for the unoptimized version.

(b) Choice of the Edge Platform: The next important decision to make is the choice of the hardware platform on which the NN model will be deployed for inferencing. This task is even more complex than the previous one as there are many parameters like the processing capability of the cores (CPU/GPU/NN Accelerator), cost, energy consumption, support for different precisions etc that needs to be carefully analyzed. *In this work, we have chosen eight different hardware platforms for our measurement study and the details are provided in the Table 2.*

(c) Choice of the software runtime libraries: The software runtime libraries act as the interface between the target hardware and the edge application. These libraries perform several optimizations like weight clustering and pruning, layer fusion, quantization of weights, input and output on the pre-trained NN models. They additionally perform hardware mapping to choose the right set of library functions for extracting maximum performance from the underlying hardware. Interaction between the software runtimes and the hardware has a direct impact on the effectiveness of the application deployment. It is thus imperative to characterize its behaviour and carefully examine the same. *In this work, we examine several state-of-the-art vendor provided runtime frameworks like ARM NN [4], PyARMNN [17], TensorRT [14], OpenVINO™ [6], Pycoral [18], Acuity [2], KSNN [13], RKNN 2 [19] and perform a thorough characterization study in Section 5.3.*

(d) Choice of the evaluation metric: It is now very clear that the final performance and the effectiveness of the deployed edge application is determined by the full stack comprising of NN model, SW runtime + OS and the hardware architecture. One of the important steps to identify the best possible solution is to choose the right metrics that can help in identifying the trade-offs available while deploying the NN based solution on the target hardware. Most of the existing benchmarking works in literature either focus only on the inference time [23] or Top-1 accuracy or both [26]. Some also consider power and energy consumption [24] as the metrics for assessing the effectiveness of the application deployment. *In this work, we discuss the trade-offs among the various first order metrics in Section 5.*

(e) Choice of synergistic utilization of heterogeneous components: As most of the hardware platforms available in the market for employing NN inference contain one or more NN accelerators, the application developer is also tasked with identifying the appropriate way to utilize the multiple hardware components together. There are many pertinent questions to consider. For example, will the individual hardware component's performance get degraded compared to its isolated execution? Will there actually be any gain due to co-execution of NN models ? If no gains are obtained, how can the available heterogeneity can still be used effectively? *In this work, we employ several techniques that can potentially utilize the available heterogeneity effectively and perform efficient NN inferencing. We perform a systematic exploration to understand the effects of these techniques on first order metrics like throughput and energy consumption.* In the subsequent sections of this paper, we delve deeper to understand the contribution of each of these choices on the actual performance of the edge application.

4 Selection through Roofline Analysis

In the quest for finding the right choices for deployment, the first step is to perform the roofline analysis for the candidate edge platforms. Roofline modeling [50] is a widely accepted performance analysis technique that combines the application characteristics with the hardware capabilities to determine the maximum attainable performance of the application on the underlying hardware. It helps in identifying whether the application's performance is limited by the compute capability or memory bandwidth of the underlying hardware.

Construction of the roofline : It involves characterizing the hardware in terms of its peak compute performance in GFLOPS (floating point operations per second) and memory performance in terms of peak sustainable bandwidth (measured in GB/s) and characterizing the application in terms of Operational Intensity (FLOPS/byte).

Operational Intensity (OI): It is an application specific parameter and is measured as the ratio of number of floating point operations performed per byte of data obtained from the main memory. Roofline bounds the maximum achievable performance of the application (measured in GFLOPS/s) as the minimum of either peak compute performance or the product of arithmetic intensity and peak bandwidth. In this work, for constructing the theoretical roofline, we made use of the hardware specifications from the data sheets (refer to Table 2) to compute the peak performance and peak bandwidth.

Peak compute capability : It is computed as a function of *no. of processor cores, operation width, SIMD width, SIMD units, no. of floating point operations per clock cycle* and the *processor frequency*.

Peak bandwidth: It is computed as a function of *memory frequency, number of channels* and *channel size*.

OI computation: To compute the operational intensity of the NN models mentioned in Section 3, we require the total number of floating point operations and total bytes of data required for computations. We made use of the inbuilt functions of tensorflow profiler to obtain the total number of floating point operations of the NN models and used the weight parameters of the NN model (refer to Table 1) to estimate the bytes of data required from the main memory in all the computations. Table 1 also provides the computed arithmetic intensities of various NN models for three different precisions.

Hardware Platform	CPU	GPU	NN-ACC	Memory	Storage	Cost
❶ Jetson Xavier AGX [7]	2.26GHz 8-core Carmel ARM v8.2	1.37GHz 512-core Volta	2x NVDLA v1	64GB 256-bit LPDDR4x	32GB eMMC	$999
❷ Jetson Xavier NX [9]	1.4GHz 6-core Carmel ARM v8.2	1.1GHz 384-core Volta	2x NVDLA v1	16GB 128-bit LPDDR4x	64GB MicroSD	$479
❸ Jetson TX2 [10]	2GHz 2-core Denver, 2GHz 4-core Cortex A57	1.3GHz 256-core Pascal	-	8GB 128-bit LPDDR4	32GB eMMC	$479
❹ Khadas VIM 3 [12]	2.2GHz 4-core Cortex-A73, 1.8 GHz 2-core Cortex-A53	800MHz Mali-G52 MP4	5 TOPS Amlogic NPU	4GB LPDDR4	64GB MicroSD	$159.90
❺ Jetson Nano [8]	1.43GHz 4-core Cortex A57	950MHz 128-core Maxwell	-	4GB 64-bit LPDDR4	16GB eMMC	$129
❻ Odorid H2 [15]	2.3GHz 4-core Intel Celeron J4105	750MHz Intel UHD Graphics 600	-	16GB DDR4	64GB eMMC	$119
❼ Intel NCS2 [5]	-	-	Myriad X VPU	500MB LPDDR	-	$110
❽ Odroid M1 [16]	2GHz 4-core Cortex-A55	650MHz Mali-G52 MP2	0.8 TOPS Rockchip NPU	8GB LPDDR4	64GB MicroSD	$95.50

Table 2. Details of the target edge devices along with specifications used for employing NN inference

Observations and Analysis: Figure 1 provides the visual representation of the constructed roofline.

`Relative Ordering:` In Figure 1(a), we observe that the Jetson TX2 CPU (2 core Denver + 4 core Cortex A57) has a better performance than 6 core Carmel CPU of Jetson NX. Similarly the, 4 core Cortex A73 of VIM3 has a better performance than the 4 core Cortex A55 of Odroid M1. Closely observing Figure 1(b), we find that for the case of FP16 precision, for workloads with **OI** $\leq$ 2.0, TX2 GPU performs better than NCS2 VPU while for workloads with **OI** $>$ 2.0, NCS2 gives $\approx$ 2.4x better performance. This is because the peak compute performance of NCS2 is 3x $>$ that of TX2 while the memory bandwidth of NCS2 is only 0.21x of TX2. Among the GPU and NPU present in the M1 board, the NPU is more powerful than the GPU but the NPU supports only INT8 precision, thus M1 NPU is used as a baseline for the INT8 while the M1 GPU is used as the baseline for FP16.

Roofline based Performance Ordering:
CPU (FP32): AGX (2.27) > TX2 (1.5) > H2 (1.15) > VIM3_Big (1.1) > NX (1.05) > M1 (1) > Nano(0.715) > VIM3_little (0.45)
GPUs and Accelerators (FP16) : AGX (117.08) > NX (55.17) > NCS2 (23.84), TX2 (16.02) > Nano (5.69) > H2 (3.47) > NVDLA (3.03) > VIM3 (1.73) > M1 (1)
INT8: AGX (24.3) > NX (11.64) > VIM3_NPU (5.54) > TX2 (3.33) > Nano (1.18) > M1_NPU (1) > VIM3_GPU (0.36) > M1_GPU (0.21).
The numbers in the brackets denote the relative performance improvement obtained (averaged across workloads) in comparison to M1.

Cost based Ordering: AGX (10.46) > NX (5.01) > TX2 (5.01) > VIM3 (1.66) > Nano (1.35)> H2 (1.25) > NCS2(1.15) > M1 (1). The numbers in the brackets denote the relative cost in comparison to Odroid-M1.

5 Performance analysis through empirical measurements

In the previous step, the NN model performance on a hardware component was estimated purely based on the data provided in the specifications. In actual application deployment, there are significant deviations from these estimates, due to several limitations of the roofline model. ❶ Theoretical roofline doesn't account the effect of caches in its performance estimation. It is well known that the caches improve the memory performance and thus there is a deviation from the estimated value. ❷ Roofline gives per hardware component wise (CPU/GPU/NPU/VPU) performance estimate. In reality, GPU and NN accelerators require the support of CPUs for the execution of the NN model and cannot work stand-alone. Thus the end-end performance of the NN model depends on the compute capability of the CPU as well. A more powerful NN accelerator coupled with a very less powerful CPU may result in worse performance than a moderately powerful NN accelerator with a more powerful CPU. The performance coupling between the CPU and the GPU/NN-accelerator cannot be directly captured using the roofline. ❸ In most hardware platforms it is observed that the actual sustainable component wise peak bandwidth is much lower than the peak bandwidth specified in the datasheets[49]. ❹ Trade-offs between various first order metrics like accuracy, energy consumption, end-to-end latency cannot be obtained without actual execution of the NN model on the hardware. ❺ The performance impact due to the choice of software runtime cannot also be ascertained without actual execution. We thus perform systematic practical measurements on the target hardware platforms and compare the observed results with the theoretical roofline observations and fine-tune our choices. *Below, we provide the details of the experimental setup used in this work for making practical measurements on the target hardware platforms.*

5.1 Experimental Setup

Our input dataset consists of images from the Imagenet validation dataset. Each image is of size 224 $\times$ 224 with 3 input channels (RGB). For each of the hardware component mentioned in Table 2, we executed all the DNN models listed in Table 1 with our input dataset. Since Intel NCS2 neural stick cannot work standalone, we have plugged NCS2 to Odroid-H2 and use NCS2 as an off-chip neural network accelerator for the Odroid-H2 platform. We use Linux based Ubuntu OS in all our hardware platforms.

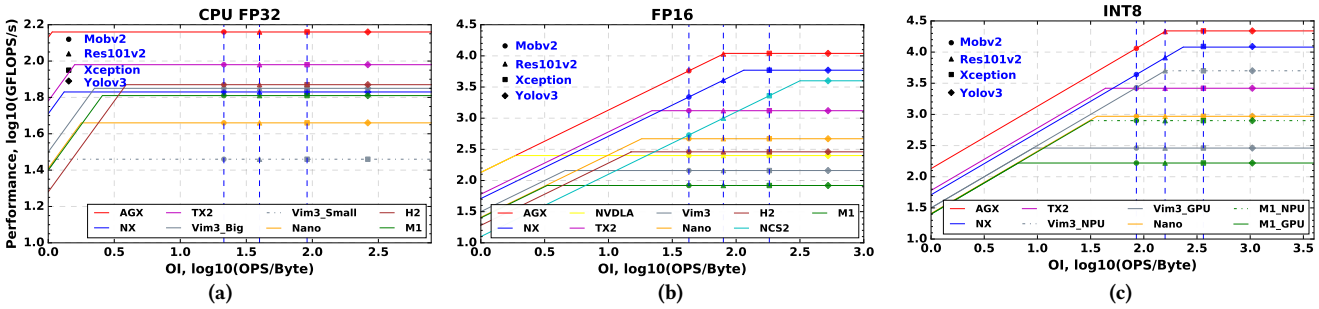


Figure 1. Roofline Model for CPU, GPU and NN accelerators

5.2 Hardware performance analysis with various metrics

We first start with the **`Inference Time`** metric. It denotes the *time taken for performing only the inference operation on a single input image*. Figure 2 shows the normalized inference time performance obtained for the execution of NN models on various processing elements of the target platforms. As mentioned earlier, Odroid-M1 is the lowest priced platform across all the target platforms and is taken as the baseline. CPUs are ubiquitous and support FP32 precision while most GPUs and NN accelerators support FP16 or INT8 precisions. Thus we use FP32 precision for the comparison of the CPUs and FP16 and INT8 precisions for comparing the GPUs and NN accelerators.

Figure 2(a) shows the relative performance of various CPUs (for FP32 precision) while figures 2(b) and (c) shows the relative performance of GPUs and NN accelerators (for FP16 and INT8 precisions) available in the target platforms when executing the NN models described in Table 1. The observations in figure 2 indicate that the

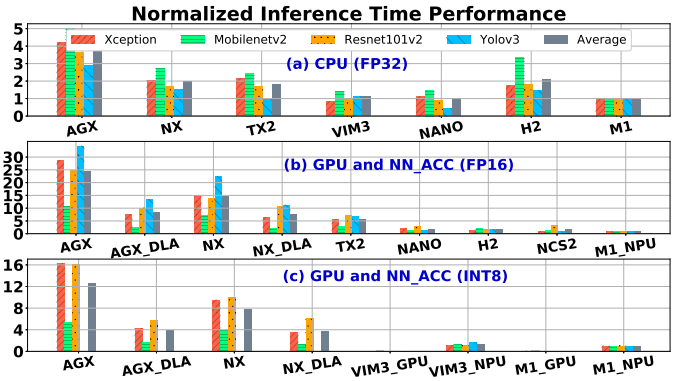


Figure 2. Normalized Inference Time performance for Xception, MobileNet-V2, ResNet-101 and Yolov3 workloads on various target platforms. Yolov3 with INT8 precision was not supported on Nvidia AGX and NX platforms.

relative performance ordering of target platforms, as determined through actual measurements, exhibits deviations from the performance ordering obtained via roofline analysis. *The ARM Carmel CPU and 512-core Volta GPU present in the AGX platform is found to be the best performing CPU and GPU in terms of throughput (considering only the inference time) with 10.69× increase in cost compared to the baseline.*

Inference time based Performance Ordering :

CPU (FP32): AGX (3.94) > H2 (2.08) > NX > (1.98) > TX2 (1.8) > VIM3_Big (1.1) > M1 (1) > Nano(0.98)
GPUs and Accelerators (FP16): AGX (24.61) > NX (14.74)> AGX_DLA(8.45)> NX_DLA(7.62)> TX2 (5.64)> NANO (1.89)> H2 (1.74) > NCS2 (1.64) > M1 (1)
INT8: AGX (12.61) > NX (7.81) > AGX_DLA(3.89) > NX_DLA(3.69)> VIM3_NPU (1.29) > M1_NPU (1) > M1_GPU (0.089) > VIM3_GPU (0.057)

Next, we analyse the **`Total Time`** metric. The total time metric encapsulates the end-end latency, encompassing the sequential steps of image loading, pre-processing, inference, and post-processing, required to classify an input image.

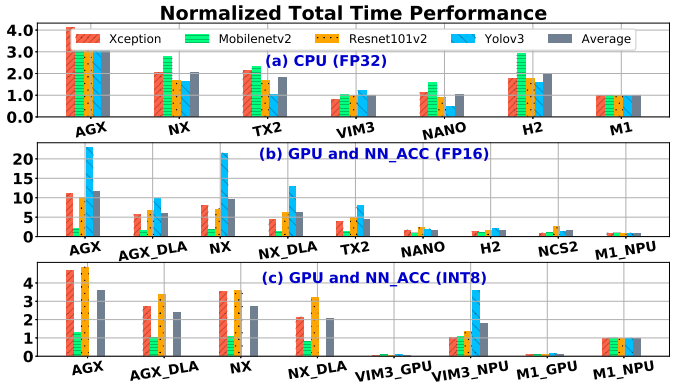


Figure 3. Normalized Total Time performance for Xception, MobileNet-V2, ResNet-101 and Yolov3 workloads on various target platforms. Yolov3 with INT8 precision was not supported on Nvidia AGX and NX platforms.

For the same set of GPUs and NN accelerators, when the metric considered is **`end-to-end latency (Total time)`** instead of **`inference time`**, the variation in the relative normalized performance changes and is much lesser than the performance reported through the inference time metric. As observed, the relative performance improvement of AGX GPU drops down to 11.64× from 24.61× and similar trends can be be observed from figures 2 and 3 for other GPUs and accelerators as well. End-to-End latency encompasses the image loading time, pre processing and post processing time in addition to the inference time. All the operations except the inference operation are executed on the CPU. The significant reduction in the relative performance degradation can be attributed to the fact that even though the GPU/accelerator performs the inference operation very efficiently, the driving CPU becomes the

bottleneck and thus there is a reduction in the final performance. Another example is the VIM3 platform with big.Little asymmetric CPU cores, the total time taken for a NN inference on GPU with little core as the supporting CPU is 1.12× more than when using big core as the supporting CPU. When using NPU accelerator, this effect becomes more pronounced and increase in total time on an average is 1.3× higher. `Thus the impact of driving CPUs potential should also be accounted for quantifying the achieved performance and the end-to-end latency is an appropriate metric to capture this impact.`

Another factor that can affect the performance are the `software run time libraries`. Suboptimal run time library implementations can hinder the performance improvement. Among the various runtime libraries used in this study, ARM NNs GPU implementation provides abysmal performance for INT8 quantization. Figure 3(c) clearly depicts this impact. We observe that when compared to the baseline, VIM3 and M1 GPUs are 12.5× and 7.8× slower.

Total time based Performance Ordering :

CPU (FP32): AGX (3.64) > NX (2.04) > H2 > (2.02) > TX2 (1.8) > NANO (1.03) > VIM3_Big (1.01) > M1 (1)
GPUs and Accelerators (FP16): AGX (11.64) > NX (9.66)> AGX_DLA(4.69)> TX2 (4.62) > NX_DLA (4.03)> NANO (1.76) > H2 (1.6) > NCS2 (1.56) > M1_NPU (1)
INT8: AGX (3.62) > NX (2.75) > AGX_DLA(2.38) > NX_DLA(2.08)> VIM3_NPU (1.78) > M1_NPU (1) > M1_GPU (0.13) > VIM3_GPU (0.08)

Next we analyze the **Energy Consumption** metric. Figure 4 shows the energy consumption of various workloads on the target platforms. We observe that NX is the best performing platform in terms of energy consumption when using GPU or NN accelerator to employ inference. Across all the target workloads, the energy consumed on NX GPU is on average 35.5% lesser than the baseline(M1_NPU). As mentioned earlier, due to poor INT8 GPU implementations of the ARM NN framework used in VIM3 and Odroid M1, the energy consumed by the GPUs of VIM3 and M1 on an average is 22.84x and 9.54x higher than the baseline. Measurement results indicate that across all the target platforms, employing inference using multicore CPU always result in higher energy consumption than using GPU or NN-accelerator except for the case of inference using ARM NN framework with INT8 quantization, the multicore CPU consumes less energy (1.6-2.7x) than its GPU counterpart. The NVDLA accelerator present in the Nvidia's AGX and NX platforms is not capable of executing all the layers of the NN model. It falls back to the GPU to execute the unsupported layers. We observe that when we use NVDLA to employ inference the energy consumed is ≈ 1.3x more than when using only GPU to employ the inference on the same platform.

Energy consumption based Ordering :

CPU (FP32): AGX (0.78) < NX (0.91) < M1 (1) < TX2 < (1.14) < H2 (1.18) < NANO (1.2) < VIM3_Big (1.35)
GPUs and Accelerators (FP16): NX (0.36) < NX_DLA (0.45) < AGX (0.64) < AGX_DLA(0.91)< TX2 (0.86) < NANO (0.89)< M1_NPU (1) < NCS2 (1.16) < H2 (2.47)
INT8: NX (0.74) < NX_DLA (0.9) < M1_NPU (1) < AGX(1.39) < AGX_DLA(1.9) < VIM3_NPU(1.18) < M1_GPU (9.54) < VIM3_GPU (22.84)

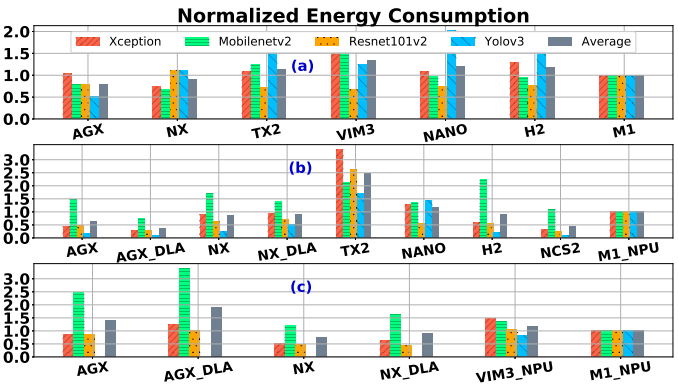


Figure 4. Energy Consumption for Xception, MobileNet-V2, ResNet-101 and Yolov3 workloads on various target platforms. Yolov3 with INT8 precision was not supported on Nvidia AGX and NX platforms.

`Average Temperature` is another closely related metric. Most of the edge devices are small form factor devices with minimal cooling capability and thus continuous usage can lead to rising platform temperatures. Higher temperatures reduce the lifetime of the device and thus it is essential to consider hardware platforms having better thermal management solution. Figure 5 shows the the relative average temperature of all the platforms comparison to the average temperature of the baseline. We can clearly observe from figure 5 that Odroid-H2-NCS platform performs poorly in terms of thermal management and the average temperature can be as high as 85 °C while for the same workload, AGX, NX, and Odorid-M1 perform well and are ≈ 35-40 °C lesser than that of H2-NCS. It can also be seen that average temperature in general is less (1-2 °C) when using NN-accelerator for employing inference.

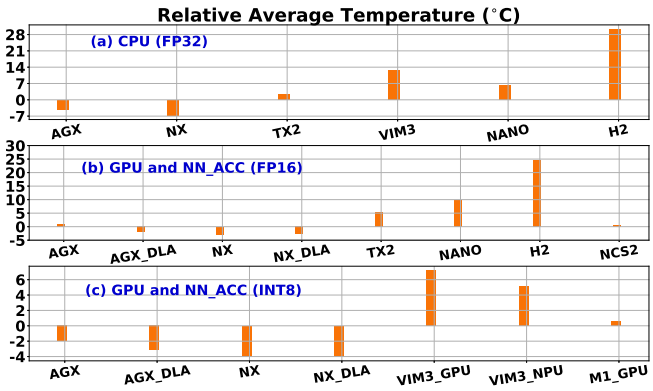


Figure 5. Relative Average Temperature averaged across workloads in comparison to the baseline(Odroid-M1) on various target platforms.

Based on the above first order metrics, we derive and propose a new metric called **`Energy-Delay-Cost Product (EDCP)`** to measure the cost-effectiveness. EDCP is computed as the product of energy consumption, end-end latency and the cost of the target platform. Lower the EDCP, better the cost-effectiveness of the target platform. We provide below the relative EDCP (averaged across the workloads) ordering of the target platforms. For comparison, we use the EDCP obtained with fastest processor available in the

target platform. We observe that Odroid-M1 platform performs best in terms of EDCP.

EDCP based Ordering :
M1 (1) < VIM3 (1.92) < NX (2.32) < NANO (4.93) < NCS2 (6.01) < AGX (7.86) < TX2 (11.13) < H2 (15.38)

5.3 Software Stack performance analysis

There can be more than one software framework available to run NN inferences on a given hardware platform. These software stacks do **(a) model compression:** Fusion of neural network layers, pruning of weights based on magnitude, quantization of 32 bit floating point weights, are typically part of these optimization pipelines. Table 3 shows the size of different NN models when a framework compresses the original NN model from 32 bit floating point precision, to 16 bit floating point or 8 bit integer. The stacks also do **(b) hardware mapping:** Mapping of the optimized neural network functions to efficient kernels for the accelerator hardware, is also done by the software stack, e.g. using CuBlas or CuDNN kernels for inferences on NVIDIA GPU running CUDA code, or using Intel MKL linear algebra functions for faster inferences on Intel processor based boards.

NN Models	TensorRT (MB)		Openvino (MB)		
	FP32	FP16	INT8	FP32	FP16
MobilenetV2	14.6	7.5	4.3	14.2	7
Resnet-101	280.4	90	46.6	178.5	89.1
Xception	93	47.1	25.8	91.5	45.7

Table 3. NN model size with vendor-provided SW framework

The software frameworks are generally of two kinds: **(a) generic frameworks:** Software giants like Google offer TFLite for running ML inferences on ARM based Android smartphones and other edhe devices. AutoTVM and AutoScheduler are similar generic software stacks from Apache supported Open Source community. The other category is **(b) hardware specific frameworks:** Typically, the hardware vendors provide a software stack for their product. For example, ARM has developed the ARMNN C++ library, and PyARMNN Python wrapper on ARMNN. TensorRT is the recommended software stack from NVIDIA to run NN inferences on the NVIDIA GPU cores. Acuity toolkit is a software given by NPU processor vendor, for VIM3 board NPU. KSNN is another software framework given by the hardware vendor building the VIM3 embedded board itself, integrating NPU with CPU cores and other peripherals.

Thus once an edge application developer chooses an embedded platform to balance the cost-performance-energy-thermal trade-offs, he might be left with a secondary choice in terms of which software framework to use. We empirically examine this secondary choice next, and its implications on the various metrics, using different software stacks for a given platform.

Effect of SW framework choice on performance metrics for a given hardware platform

(a) Throughput gap across software frameworks for a given hardware platform: Figure 6 shows the throughput for two NNs, Densenet and Xception, on a particular edge hardware platform Odroid M1, which has both CPU and GPU cores from ARM. As mentioned above, ARM offers a C++ inference library ARMNN and a Python wrapper on top of it, namely PyARMNN, as software frameworks for fast NN inferences. Google's TFLite framework can also be used on this platform, for running CPU inferences, not GPU. ARMNN and TFLite can use one to four threads on the four CPU cores, whereas PyARMNN supports only single CPU thread.

ARMNN is thus the best among the three compared software frameworks, in terms of supporting both CPU and GPU, and also multi-threaded inferences on CPUs. The throughput achieved by ARMNN and PyARMNN are the same, and higher than TfLite, for both NN models. Thus different software stacks give different throughputs on the same processor core for a given hardware platform, and the hardware vendor's stack outperforms the generic software stack TfLite.

(b) Do vendor provided framework give highest throughput for all platforms? We next examine whether this observation that *hardware vendors give the most optimal software stack for their platforms/processors, in terms of throughput*, that held true for ARMNN stack on Odroid M1 board above, is true in general. We examine the same two NN models (DenseNet and Xception) and the same three software frameworks (ARMNN, PyARMNN and TfLite) now on the VIM3 board. VIM3 board also comprises ARM CPU and GPU cores. The throughput values are shown in Figure 7. Here ARMNN still has the advantage of supporting both CPU and GPU, and also multi-threaded programs on CPU, as in Odroid. However, on VIM3, TfLite gives much higher throughput for both NN models, compared to both ARMNN and PyARMNN, which is in contrast to what we observed in Odroid M1 earlier.

There is thus no generalizable observation that hardware vendors such as ARM know their hardware platforms the best, and therefore can extract maximal performance from the hardware. Generic software stacks like TfLite offered by software companies like Google can outperform hardware vendor stacks, as seen here for VIM3 platform.

The Odroid M1 and VIM3 boards have less powerful processors, that can give at most 6-7 FPS with any software framework. The performance gap across software stacks on these less powerful boards is therefore only of 1-2 FPS. More dramatic differences can be seen in powerful edge platforms like NVIDIA AGX. Figure 8 first plot shows Xception model throughput for NVIDIA stack TensorRT and a generic stack AutoScheduler. The throughput gap is 400 FPS (AutoScheduler) vs. 150 FPS (TensorRT) for FP16, and 600 FPS (AutoScheduler) vs. 200 FPS (TensorRT) for INT8. While TensorRT is by default included in the software packages, when an application developer buys an NVIDIA edge platform, he should really examine open source software alternatives like AutoScheduler, to not miss a possible throughput jump of several hundred FPS!

(c) Are frameworks from hardware vendors optimal for thermal and energy constraints? A throughput jump of several hundred FPS, as seen on NVIDIA AGX GPU using AutoScheduler software stack instead of NVIDIA TensorRT, might be suspected by some application developers to have significant trade-off in terms of much higher temperature rise and energy usage. The second and third plots in Figure 8 removes this suspicion, showing AutoScheduler gives only slightly higher or the same temperature as TensorRT, whereas using significantly less energy per frame. AutoScheduler is really a very well implemented software library for NN model optimization and mapping the NN operators to CUDA kernel functions, and is significantly better than the software stacks developed by hardware vendors for all performance metrics.

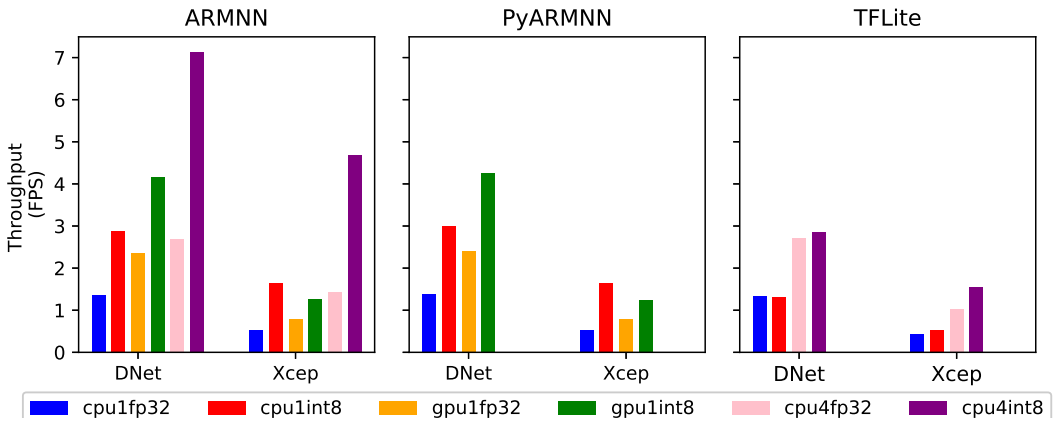


Figure 6. ARMNN and PyARMNN gives *higher* throughput than TFLite on the Odroid M1 board's CPU.

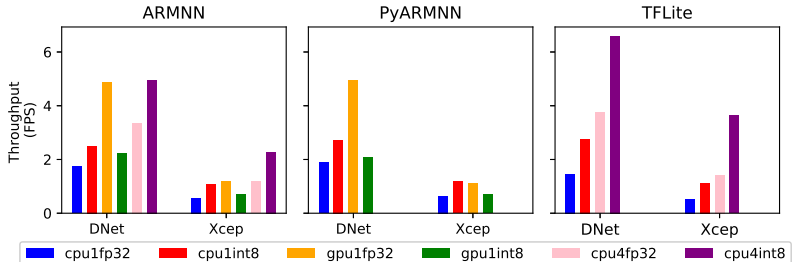


Figure 7. ARMNN and PyARMNN gives *lower* throughput than TFLite on the VIM3 board's CPU.

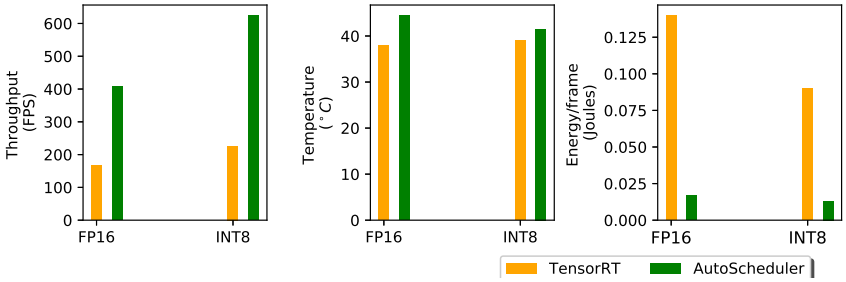


Figure 8. TensorRT gives *lower* throughput, comparable temperature and *higher* energy than AutoScheduler on the NVIDIA AGX GPU. Hardware vendor stack is *worse* than generic stack in both throughput and energy.

(d) How does various frameworks perform w.r.t Accuracy? The TensorRT library that powers the NN execution on the Nvidia GPUs provides ≈81.6% accuracy for the Xception model with FP16 precision while the Intel's OpenVINO provides ≈80.92%. ARM NN provides a close ≈79.17% accuracy for FP32.

Precision vs Accuracy: Across all the runtimes studied, for Xception model, accuracy loss is insignificant (≈0.02-0.04%) when using reduced precision (FP16) instead of FP32. While using INT8 quantization, the drop in the accuracy is in the range of 2-2.4% across most runtimes. However, for tflite models running on Nvidia's CPUs, the accuracy loss when moving from FP16 to INT8 is quite high (≈15%). The accuracy provided by accelerator runtimes like KSNN and RKNN is $\tilde{2}$-3% lesser than the TensorRT and OpenVINO™. Similar trends are observed for MobileNet-V2 and ResNet-101 workloads as well. The accuracy values for MobileNet-V2 are in the range of 69.8-74.2% (for FP16) and 63.8-72.7% (for INT8). For ResNet-101, it is in range of 71.2-75.2% (for FP16) and 69.3-74.9%(for INT8).

6 Concurrent execution on Heterogeneous Co-processors

Most of the current generation edge devices contain multiple generic processors like CPU and GPU and one or more specialized NN accelerators like DLA, VPU and NPU. Section 5 and Section 5.3 empirically measure the performance of each of these processing elements in isolation (solo), i.e. running NN inference on single processor/accelerator at a time. This leaves other processors under-utilized. In this section, we examine concurrent execution of NN inferences, using more than one processing element and analyse how different target platforms support concurrent execution. The purpose of this study is to understand how effectively the resources in the platform can be leveraged under practical constraints.

6.1 Experimental setup for concurrency analyses

The GPU and DLA processing elements in Nvidia's AGX and NX are both provided by NVIDIA. Similarly, the CPU, GPU processors on Odroid-H2 as well as the NCS-2 accelerator are provided by same vendor, Intel. These two boards are thus **homogeneous**

vendor devices. In contrast, Khadas VIM3 with ARM big.LITTLE CPU architecture and Mali GPU, have CPU and GPU from the same vendor ARM, but its NPU accelerator is provided by a different vendor VeriSilicon. It is thus a **heterogeneous vendor device**. The software stacks usable on an edge platform depends on whether the processor cores are from homogeneous vendor or heterogeneous vendor. Nvidia platforms (AGX and NX) can use the same software framework **TensorRT** while Intel's Odroid-H2 and NCS2 can use **OpenVINO™** to run NN inferences on all the available co-processors. VIM3, however, has to use ARMNN/PyARMNN for NN inferences on the ARM CPU/GPU cores and Acuity/KSNN framework for employing NN inferences on the NPU. For this concurrency study, we choose (a) Intel Odroid-H2 along with Neural Compute Stick2(NCS2) *(connected via USB3.0)* and (b) Khadas VIM3, to have a mix of homogeneous and heterogeneous platforms and are of similar price range.

6.2 NN performance through concurrent execution:

It is imperative that when more than one processor is employed for NN inference, higher NN inference performance is expected compared to solo runs. We now examine whether this expectation of higher performance is met, using two different concurrency strategies across the co-processors: **(a) data parallel or data-wise** workload allocation and (**(b) task parallel or NN layer-wise** workload allocation.

6.2.1 Concurrent NN inferences using Data-wise Workload Allocation (DWA)

An intuitive way of splitting input data among the processors is as follows. *If the processor A is x times faster than processor B in solo runs, the data is divided in the ratio of x : 1 to A and B respectively when used together.* OpenVINO's MULTI plugin splits workload across the available co- processors in the platform following this heuristic. E.g. ResNet-101 concurrent inferences using both CPU and GPU on H2 will have 5.17× more data for GPU, while VPU-CPU concurrent inferences will have 3.99 × more data for VPU. On the heterogeneous VIM3 board, due to unavailability of unified framework, we manually perform this workload split among the CPU, GPU and NPU, using a combination of ARMNN and KSNN frameworks. We make the following observations:

(a) Odroid-H2 + NCS2 : DWA concurrency reduces inference latency but increases energy consumption. Figure 9 shows the results of DWA concurrency on Odroid-H2. All three NN models show improvement in total run-time compared to best performer in solo runs. Xception achieves 42.8% lower run-time on GPU-VPU DWA compared to GPU solo (fastest solo). Using all the available processors (CPU, GPU & VPU) the total run-time improvement is 35.14% while CPU-GPU and CPU-VPU do not achieve any improvement (-0.1% and -4.7%). On MobileNet-V2 similar trends can be observed. On ResNet-101 also, VPU-GPU achieve 35.7% improvement while combination using all the processors achieve 30.44% improvement. Combinations involving CPU do not achieve any notable improvement, instead CPU-GPU report a degradation of 79.6%.

DWA concurrency increases energy consumption on Odroid-H2 (Figure 9), as more processors working parallely draw more power. The GPU-VPU has the best trade-off for all the models in-terms of total run-time improvement and energy utilization. E.g., MobileNet-V2 uses 35.2% more energy than VPU solo while achieving 44.5% improvement in run-time. CPU-VPU also shows similar increase in

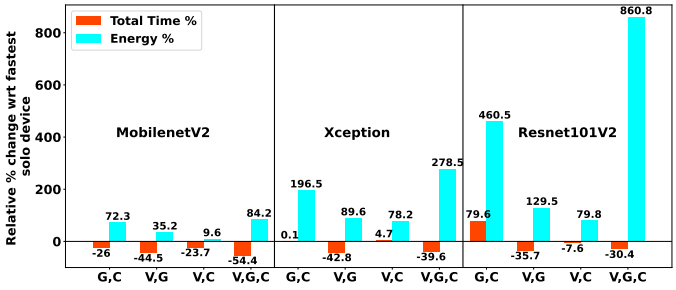
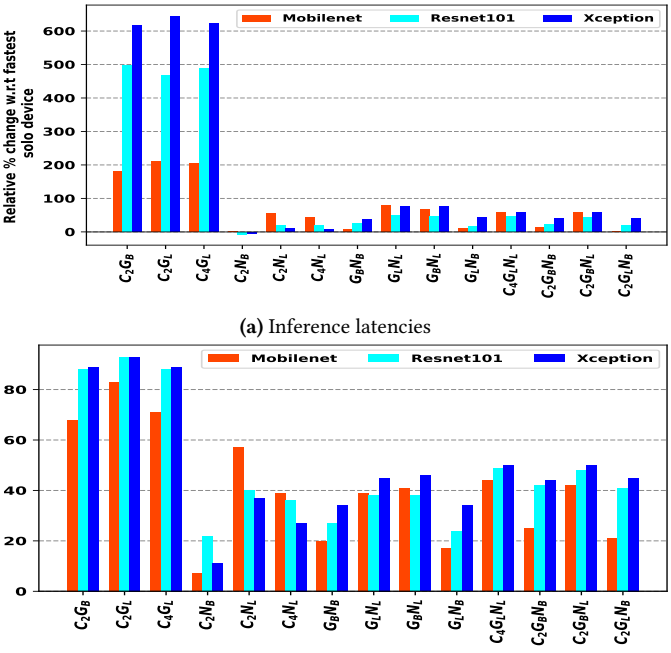


Figure 9. Effect of DWA inference on Energy & run-time for Odroid-H2. The values are plotted w.r.t most efficient inference in solo run/model. The negative values indicate the % improvement and positive values show degradation. *x-axis represents the processors involved in the split, i.e. **C**PU, **G**PU & **V**PU.*

energy utilization, but improvement in run-time is very negligible. For all other DWA combinations i.e. CPU-GPU and CPU-GPU-VPU the improvement in run-time is overshadowed by increase in energy utilization, thus making them less likely choice for DWA execution. E.g in CPU-GPU-VPU the energy utilization for ResNet-101 increases by 860.8% with only 30.44% improvement in total run-time.

(b) VIM3: DWA concurrency increases both inference latency and energy. In Figure 10a, we can observe that VIM3



(a) Inference latencies

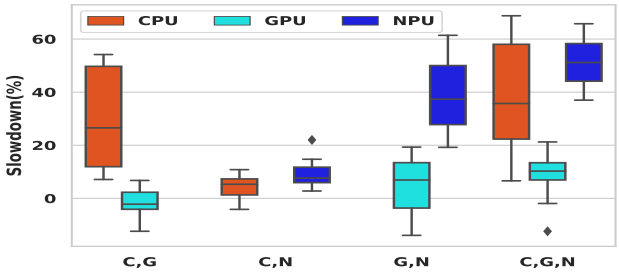
(b) Inference energy

Figure 10. Inference latency and energy in Khadas-VIM3 under DWA concurrency. Axes are similar to Figure 9. The number with each processor are the threads used. **B** & **L** represent the Big and Little CPU cores used.

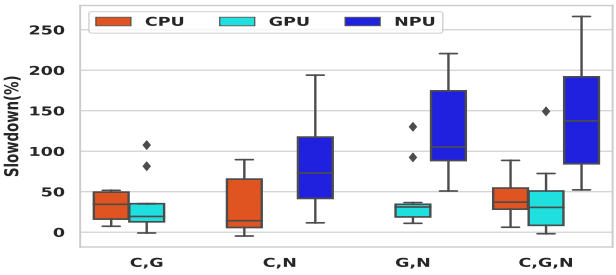
does not show inference time improvement on any of the NN model workloads compared to the solo best performance, in contrast to Odroid. Xception & ResNet-101 show a meager run-time improvement of 4.8% & 7.4% respectively on CPU (2 threads)-NPU DWA. The remaining combinations show huge degradations in runtime. E.g. for Xception using all the available processors, the run-time

degrades by 58.4%. Similarly on MobileNet-V2, the degradation in total run-time ranges from 1.8% to 205%, while as on ResNet-101 the degradation reaches up-to 497%. Energy numbers are also not promising for DWA concurrency on VIM3 (Fig. 10b). ResNet-101 shows the degradation of 21.9% while Xception & MobileNet degrades at 10.8% and 6.9% respectively, w.r.t their solo performance on NPU. Combinations involving NPU consume less energy, as NPU is highly energy efficient and gets maximum share of input data for inference as it is also the fastest among the three processors.

(c) Why does DWA concurrency degrade both latency and energy on VIM3? DWA concurrency affects processor performance, as shared resources like DRAM, cache and buses cause contention among the different processors. On VIM3, all three processing elements CPU, GPU and NPU are on the same chip, each having very less internal memory and shared access to DRAM, causing significant contention during concurrent processing.



(a) Execution time/inference



(b) Pre-processing Time

Figure 11. Degradation of individual processors in various DWA schemes on VIM3. For every processor in any DWA scheme, values are plotted w.r.t its performance in solo runs. x-axis represents processors

In Figure 11a, the minimum degradation of execution time is reported in CPU-NPU combination where CPU shows average degradation of 3.7% and NPU 9.6%. In all other DWA schemes involving CPU, the degradation ranges from 28% to 38%, making CPU the least preferred choice to be used in co-processor execution. NPU being the fastest processor in solo inference, still gets degraded at similar rate as CPU. Figure 11b shows that NPU is also worst effected in pre-processing time, being 87% to 138% slower than NPU solo. This possibly results from the fast rate at which the NPU accelerator requires pre-processed inputs, increasing resource contention.

(d) But why does resource contention not degrade NN inference latency on Odroid-H2? Individual processor performance gets degraded compared to solo execution, also in Odroid-H2 concurrency (Table 4). In CPU-GPU DWA combination the GPU is most penalized at 15% for MobileNet-V2, 25.34% for ResNet-101 and 19.5% for Xception. However, the maximum degradation of VPU is just 0.7% in GPU-VPU combination for Xception. The negligible degradation in VPU run-time performance is due to the fact that VPU is off-chip and has its own internal memory that can hold a fairly large NN model like ResNet-101 or Xception, thereby not requiring frequent memory accesses like in case of CPU or GPU.

6.2.2 Concurrent NN inferences using Layer Wise Workload Allocation (LWA) In LWA, different layers of NN model are executed on different co-processors such that workload division takes place at NN layer level. We experiment with several schemes to split NN layers among processors. ❶ **LWA-default scheme**, where all layers supported on the fastest processor are run there, and slower processors are used for unsupported layers. ❷ **LWA-R scheme**, where the layer of NN model having lowest runtime on any processor is mapped on that processor. ❸ **LWA-P scheme**, where the top-n layers of the NN model having the highest number of parameters, or the compute subgraph with highest number of parameters are mapped on to the fastest processor. ❹ **LWA-C scheme**, the top-n layers of the NN model having the highest compute (GFlops) are mapped on to the fastest processor.

(a) LWA is slower than solo inference on the fastest processor Table 5 shows the average inference time (AIT) and end-to-end time (E2E) for solo runs and LWA-R inference on Odroid-H2. It must be noted that for MobileNet-V2 the LWA-R inference performs worse than all solo runs, i.e. 52.8% lower than CPU (solo worst). For ResNet and Xception the improvement in AIT can only be seen with respect to CPU (slowest solo). It results in performance degradation of 50.3% and 69.4% for ResNet-101 and Xception of AIT when compared to the fastest solo device.

(b) Why does LWA degrade inference and end-to-end latencies? Inter processor communication can be a major limiting factor in LWA. As seen from Table 5, the inter processor communication takes place 25 times in MobileNet-V2 i.e. 25 times the subsequent layers are mapped on different processors. Similarly for Xception and ResNet-101 the inter processor communication takes place 13 and 63 times respectively. In terms of run-time it acts as a limiting factor as the intermediate layer outputs are to be transferred back and forth between the processors. The overhead for this transfer is non-trivial and amounts to about 2.5% to 25% of end to end time.

(c) Is subgraph based LWA, that reduces transfers across processors, faster? Table 6 shows inference time for LWA-P scheme on Odroid-H2. In this scheme the top-n layers with highest parameters are mapped to the secondary device, and other layers of the model are run on the primary device. E.g. if the LWA-P is used with CPU-GPU, the top-n parameter layers will be mapped to GPU *(secondary device)* while CPU *(primary)* will run all other layers. We compare this with the sub-graph allocation of highest parameters on secondary processor. A sub-graph is chosen as n continuous layers from the network having highest parameters or any other metric. Table 6 shows values for $n = 5$. In all cases, the mapping of top-1 sub-graph performs better than top-5 parameter layer mapping. Top-5 parameters layers are random in NN compute graph, and results in maximum of $n \times 2$ transfers between the primary and secondary processors. But top-1 sub-graph only involves 2 transfers between primary and secondary processors (for any n), and has much lower latency.

As observed, an edge platform like VIM3 shows no performance improvement in terms of NN inference throughput, when more than one processor are used concurrently. We therefore ask the pertinent

NN Models	CPU[2] GPU[2]		CPU[2] VPU[2]		GPU[2] VPU[2]		CPU[2] GPU[1] VPU[1]		
	CPU	GPU	CPU	VPU	GPU	VPU	CPU	GPU	VPU
MobileNet-V2	8.08	9.69	2.7	3.95	2.85	3.99	4.36	7.55	3.22
Resnet101	10.8	21.4	0.45	0.87	0.12	1.39	8.58	16.67	3.44
Xception	11.86	17.88	1.11	0.47	1.38	1.55	11.75	17.77	1.32

Table 4. Percentage degradation of inference time when run in split compared to solo run for Odroid-H2

NN Models	CPU		GPU		VPU		$Device_{min}$		
	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	Transfers
MobileNet-V2	17.83	140.02	8	100.3	12.52	117.31	33.71	295.49	25
Resnet101	327.35	1689.23	86.82	517.22	48.99	298.9	97.32	612.64	63
Xception	398.13	2053	141.97	779.62	185.43	1019.02	214.3	1173.71	13

Table 5. Run-time analysis of LWA-R scheme. CPU, GPU and VPU represent the average inference time (AIT) in *ms* and End to End time (E2E) in *s* for solo runs while as $device_{min}$ show the LWA-R execution.

NN Models	Top-5 parameter layer CPU, GPU		Top-1 subgraph CPU, GPU		Top-5 parameter layer CPU, VPU		Top-1 subgraph CPU, VPU		Top-5 parameter layer GPU, VPU		Top-1 subgraph GPU, VPU	
	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)	AIT (ms)	E2E (s)
MobileNet-V2	31.28	215.04	29.27	204.91	30.9	219.04	28.5	206.4	15.6	133	13.9	125.6
Resnet101	354.4	1840.26	345.7	1792.61	355.5	1851.6	350.4	1812.5	128.6	723.3	120.1	679.5
Xception	410.14	2118.14	406.6	2098.71	412.14	2129.4	410.9	2115.4	190.64	1030.2	180.4	979.1

Table 6. Run-time analysis of LWA-P scheme. Top-5 parameter layers represent the layers with highest no. of parameters while as Top-1 sub-graph is the sub-graph of 5 layers with highest parameters. The processor combination in row-1 is in the form of **primary, secondary**.

question, *Is there any benefit in having multiple processors in this kind of edge platform, (increasing cost and programming complexity) at no added performance improvement under concurrent execution ?* We answer this using a concrete application use case in the next section, demonstrating that heterogeneous co-processors can be useful to achieve other kind of metric trade-offs in real applications, even if a particular metric like throughput cannot be boosted.

Observations: ❶ DWA works better with off-chip co-processors (e.g. Odroid-H2+ NCS2), compared with device with all on-chip accelerators (e.g. VIM3). All on-chip processors are likely to increase the contention, thereby delaying the access to the inputs for every inference. ❷ If speed disparity across processors of a particular board is high, it is unlikely that the DWA scheme will achieve any notable benefit. This is seen in Odroid H2's slow CPU, which gave negligible improvement when combined with much faster GPU or VPU. ❸ LWA performance suffers from inter-processor communication, that degrades performance compared to solo run on the fastest processor.

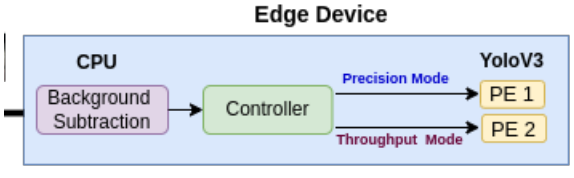
7 Application case study: Vehicle Detection on Indian Traffic Dataset

This section demonstrates a concrete use case of vehicle detection on Indian traffic dataset using intersection cameras, where heterogeneous co-processors on an edge platform, can be intelligently used by an application developer, to achieve interesting accuracy-latency trade-offs.

What is the application and associated NN task? What are the important metrics? Traffic at intersection of developing country like India is chaotic and non-lane based. There are multiple rule violations like jumping the red light, which traffic police tries to catch with intersection cameras and fine the vehicle. Detection of vehicles is the NN task, for which an object detection NN like YOLO[43] needs to be used. Figure 12a shows sample images from a dataset we have collected in collaboration with Delhi traffic authorities.

Metrics of importance depend on the traffic density, which varies with time of the day – like low density during early morning time, high density during peak office hours, and moderate density otherwise. Vehicle detection should run at high FPS during low traffic density, as vehicles are moving fast and the scene is changing quickly. During high traffic density, it is acceptable if vehicle detection is slow, as the scene hardly changes. However, the detection precision in high traffic should be high, as many objects stand in close proximity. Thus target application metrics are high FPS in low traffic density, and high precision with allowance for low FPS in high traffic density.

(a)



(b)

Figure 12. Intersection camera dataset sample images from Delhi India in (a) showing low, medium and high traffic densities. Application workflow in an edge platform like VIM3 in (b). PE denotes processing elements, e.g. NPU and GPU in VIM3.

How can heterogeneous processors on an edge device balance these metrics? Table 7 shows the execution time in seconds to process 200 images per traffic density category, throughput in frames/second and average precision, of running YOLO V3 object detection model on Khadas VIM 3 GPU and NPU processors. We see that the NPU is almost 9 times faster than the GPU, with lower execution time and higher throughput. But the GPU has very high precision even on high traffic density frames, as it uses FP32, whereas the NPU can only run INT8 quantized models. NPU INT8 model gives good precision in low traffic density, as vehicles are sparse and easier to detect. GPU can therefore be used when high precision is needed in high traffic, and NPU can be used when high throughput is needed in low traffic.

Metrics	GPU			NPU		
	High	Medium	Low	High	Medium	Low
Execution Time (s)	272.59	259.897	261.304	29.517	32.551	27.62
Throughput	0.73	0.77	0.77	6.78	6.14	7.24
AP @ 0.75	96.3	95.6	92.6	90.1	92.5	94.4

Table 7. Yolo-V3 performance for object detection on VIM3 platform; Execution Time = Number of images * Total time(primary metric) and AP refers to Average Precision. We use 200 images for each traffic density.

Detecting traffic density at run-time is a computationally simple task that can run on CPU, without using CNN, based on background subtraction computer vision algorithm. Figure 12 shows the end-to-end workflow for this application, where incoming frames are first processed by CPU for traffic density estimation. If high density is detected, the frame is passed to PE1 i.e. GPU in this case, which will take time but give very precise output. If low traffic density is detected, the frame is passed to PE2 i.e. NPU in this case, which will give prompt output, with reasonable precision.

As shown, using the lowest performance processor CPU to monitor application state (traffic density) and choosing between GPU and NPU to balance precision-latency trade-off, all the three co-processors on the VIM3 platform can be effectively leveraged for end-to-end application management.

8 Summary

In this work, we perform empirical characterization of different hardware platforms using a diverse set of NN models along with different kinds of software runtime library frameworks. We show in this article that it is important to consider the impact of driving CPU's potential for quantifying the true performance of the NN accelerator. We introduce a new metric called *EDCP* to capture the cost effectiveness of the heterogeneous platform. We also show that hardware vendors might not be the best software runtime providers for their own hardware platform . Finally we study how effectively the co-processors in the platform can be leveraged and employ different concurrency strategies for workload allocation across the co-processors. In future, we plan to examine other workloads like cryptographic workloads as platform security and data privacy are also of paramount interest. Finally, we will continue to work with field practitioners (like Traffic Police we collaborate with in Section 7), so that these deep technical insights about edge platforms and software stacks can be useful in practice.

9 Acknowledgments

We thank our reviewers for their insightful feedback that helped us to improve our paper. This work was partially supported by the Science and Engineering Research Board, Department of Science and Technology, India under grant SERB-POWER(SPG/2021/000731).

References

[1] [n. d.]. The 2022 Gartner Hype Cycle™for Artificial Intelligence (AI). https://www.gartner.com/en/articles/what-s-new-in-artificial-intelligence-from-the-2022-gartner-hype-cycle.
[2] [n. d.]. *Acuity Toolkit : https://github.com/khadas/aml_npu_sdk*.
[3] [n. d.]. AI in Edge Devices. https://www.linleygroup.com/events/agenda.php?num=49&day=1.
[4] [n. d.]. *ARMNN : https://github.com/ARM-software/armnn*.
[5] [n. d.]. Intel NCS2. https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html.
[6] [n. d.]. Intel® Distribution of OpenVINO™ Toolkit. https://github.com/openvinotoolkit/openvino.
[7] [n. d.]. Jetson Xavier AGX. https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-agx-xavier/.
[8] [n. d.]. Jetson Xavier Nano. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
[9] [n. d.]. Jetson Xavier NX. https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-xavier-nx/.
[10] [n. d.]. Jetson Xavier TX2. https://developer.nvidia.com/embedded/jetson-tx2.
[11] [n. d.]. Keras Applications. https://keras.io/api/applications/.
[12] [n. d.]. Khadas Vim3. https://www.khadas.com/vim3.
[13] [n. d.]. *KSNN : https://github.com/khadas/ksnn*.
[14] [n. d.]. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt.
[15] [n. d.]. ODROID H2. https://www.hardkernel.com/shop/odroid-h2/.
[16] [n. d.]. Odroid M1. https://www.hardkernel.com/shop/odroid-m1-with-8gbyte-ram/.
[17] [n. d.]. *PyArmNN : https://github.com/NXPmicro/pyarmnn-release*.
[18] [n. d.]. *Pycoral : https://github.com/google-coral/pycoral*.
[19] [n. d.]. *RKNN : https://github.com/rockchip-linux/rknn-toolkit*.
[20] [n. d.]. Top 5 Edge AI Trends to Watch in 2023. https://blogs.nvidia.com/blog/2022/12/19/edge-ai-trends-2023/.
[21] Hazem A. Abdelhafez, Hassan Halawa, Amr Almoallim, Amirhossein Ahmadi, Karthik Pattabiraman, and Matei Ripeanu. 2022. Characterizing Variability in Heterogeneous Edge Systems: A Methodology & Case Study. In *7th IEEE/ACM Symposium on Edge Computing, SEC 2022, Seattle, WA, USA, December 5-8, 2022*. IEEE, 107–121.
[22] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2016. Fathom: Reference workloads for modern deep learning methods. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.
[23] Mattia Antonini, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. 2019. Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things* (New York, NY, USA) *(AIChallengeIoT'19)*. Association for Computing Machinery, New York, NY, USA, 49–55. https://doi.org/10.1145/3363347.3363363
[24] Martin Arlitt, Manish Marwah, Gowtham Bellala, Amip Shah, Jeff Healey, and Ben Vandiver. 2015. IoTAbench: An Internet of Things Analytics Benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering* (Austin, Texas, USA) *(ICPE '15)*. Association for Computing Machinery, New York, NY, USA, 133–144. https://doi.org/10.1145/2668930.2688055
[25] S. Baller, A. Jindal, M. Chadha, and M. Gerndt. 2021. DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE Computer Society, 20–30. https://doi.org/10.1109/IC2E52221.2021.00016
[26] Simone Bianco, Rémi Cadène, Luigi Celona, and Paolo Napoletano. 2018. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access* 6 (2018), 64270–64277. https://doi.org/10.1109/ACCESS.2018.2877890
[27] Michaela Blott, Lisa Halder, Miriam Leeser, and Linda Doyle. 2019. Qutibench: Benchmarking neural networks on heterogeneous hardware. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15, 4 (2019), 1–38.
[28] Tianshi Chen, Yunji Chen, Marc Duranton, Qi Guo, Atif Hashmi, Mikko Lipasti, Andrew Nere, Shi Qiu, Michele Sebag, and Olivier Temam. 2012. BenchNN: On the broad potential application scope of hardware neural network accelerators. In *2012 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 36–45.
[29] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.
[30] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2019. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark.

ACM SIGOPS Operating Systems Review 53, 1 (2019), 14–25.

[31] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proceedings of the 3rd workshop on general-purpose computation on graphics processing units*. 63–74.

[32] DeepBench. 2018. DeepBench-2018. In *https://svail.github.io/DeepBench/*.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 248–255.

[34] Johann Hauswald, Yiping Kang, Michael A Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G Dreslinski, Jason Mars, and Lingjia Tang. 2015. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 27–40.

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

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

[37] Spencer Lin Jason Jackson and IBM Marcelo Sávio. [n. d.]. The edge computing advantage. https://www.ibm.com/downloads/cas/Y7RA6X93.

[38] EunJin Jeong, Jangryul Kim, Samnieng Tan, Jaeseong Lee, and Soonhoi Ha. 2021. Deep learning inference parallelization on heterogeneous processors with tensorrt. *IEEE Embedded Systems Letters* 14, 1 (2021), 15–18.

[39] Jongmin Jo, Sucheol Jeong, and Pilsung Kang. 2020. Benchmarking gpu-accelerated edge devices. In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 117–120.

[40] Duseok Kang, DongHyun Kang, Jintaek Kang, Sungjoo Yoo, and Soonhoi Ha. 2018. Joint optimization of speed, accuracy, and energy for embedded image recognition systems. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 715–720.

[41] Yu Liu, Hantian Zhang, Luyuan Zeng, Wentao Wu, and Ce Zhang. 2018. MLbench: benchmarking machine learning services against human experts. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1220–1232.

[42] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[43] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. https://doi.org/10.48550/ARXIV.1804.02767

[44] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2019. Survey and benchmarking of machine learning accelerators. In *2019 IEEE high performance extreme computing conference (HPEC)*. IEEE, 1–9.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[46] Linpeng Tang, Yida Wang, Theodore L Willke, and Kai Li. 2018. Scheduling computation graphs of deep learning models on manycore cpus. *arXiv preprint arXiv:1807.09667* (2018).

[47] Jin-Hua Tao, Zi-Dong Du, Qi Guo, Hui-Ying Lan, Lei Zhang, Sheng-Yuan Zhou, Ling-Jie Xu, Cong Liu, Hai-Feng Liu, Shan Tang, et al. 2018. Benchip: Benchmarking intelligence processors. *Journal of Computer Science and Technology* 33, 1 (2018), 1–23.

[48] Blesson Varghese, Nan Wang, David Bermbach, Cheol-Ho Hong, Eyal De Lara, Weisong Shi, and Christopher Stewart. 2021. A Survey on Edge Performance Benchmarking. *ACM Comput. Surv.* 54, 3, Article 66 (apr 2021), 33 pages. https://doi.org/10.1145/3444692

[49] Siqi Wang, Anuj Pathania, and Tulika Mitra. 2020. Neural network inference on mobile SoCs. *IEEE Design & Test* 37, 5 (2020), 50–57.

[50] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (4 2009). https://doi.org/10.1145/1498765.1498785

[51] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 392–405.

10 Appendices

Overview of prior NN Benchmark study

The detailed comparison of prior work w.r.t to the three aspects (SW frameworks, quantization and workload allocation) along with its evaluation objective is summarized below in Table 8.

NN System Benchmarks	Evaluation criteria used	Quantizatio Support	Workload Allocation	SW framework Analysis
DeepEdge Bench [25]	• Compares the performance of embedded devices like Raspberry Pi, Coral Dev borad, Jetson Nano etc in-terms of inference time and power consumption. • Shows the effect of using dedicated AI processing units on inference time versus CPU only inference.	✓	✗	✓limited
QuTiBench [27]	• Compare the variety of algorithmic optimization options particularly quantization. • Help system level designers understand the limitations and benefits of specific compute architecture.	✓	✗	✗
BenchIP [47]	• Micro bench-mark and macro bench-marks to evaluate single layer networks and large industrial scale neural networks.	✗	✗	✗
Fathom [22]	• Benchmarks NN workloads beyond CNNs and includes training as well. • Does not support heterogeneous hardware, advocates unified software.	✗	✗	✗
MLPerf [42]	• Similar to Fathom, adds applications like sentiment analysis and recommendation systems as target applications. • Provides open models allowing algorithmic optimizations facilitating improvements for specific hardware architecture.	✗	✗	
DAWN bench [30]	• Evaluates time and cost (USD) of popular deep learning training and inference workloads. • Introduces a new metric *time to accuracy (TTA).* • Exclusive ImageNet training and inference with very few evaluation metrics, like *time required for the model to obtain Top-5 accuracy of 90% or above.*	✓	✗	✗
Jongmin et al [39]	• Analysis of GPU accelerated edge devices comparing the performance of Jetson Nano with Jetson TX2, GTX 1060 (Desktop grade system) and Tesla V100 (server grade).	✗	✗	✗
Reuther et al [44]	• Comprehensive survey and benchmark of machine learning accelerators in terms performance/power trade-off for wide category of devices viz low power, embedded, autonomous, data-center systems and cards.	✓	✗	✗

Table 8. Comparison w.r.t existing NN Benchmarks.