PiQi: Partially Quantized DNN Inference on HMPSoCs

Ehsan Aghapour, Yixian Shen, Dolly Sapra, Andy D. Pimentel, and Anuj Pathania
University of Amsterdam
e.aghapour@uva.nl, y.shen@uva.nl, d.sapra@uva.nl, a.d.pimentel@uva.nl, a.pathania@uva.nl

ABSTRACT

Deep Neural Network (DNN) inference is now ubiquitous in embedded applications at the edge. State-of-the-art Heterogeneous Multi-Processors System-on-Chip (HMPSoCs) powering these applications come equipped with powerful Neural Processing Units (NPUs) that significantly outperform other inference-capable HMPSoC components – namely, the CPUs and GPUs – in terms of power consumption and performance. However, CPUs and GPUs can perform full precision inference, whereas NPUs can often only perform a quantized inference. Consequently, low-latency, low-power inference by the NPU comes at an accuracy loss due to the quantization.

DNNs consist of several heterogeneous layers. Here, we introduce the *PiQi* framework that allows DNN inference to layer-wise switch between the three inference-capable HMPSoC components, CPU, GPU, and NPU, mid-inference with minimal overhead. Consequently, *PiQi* employs the novel idea of partially quantized DNN inference on HMPSoCs. However, different DNN layers experience different power-performance gains while projecting different accuracy losses on quantization. Therefore, we provide within *PiQi* a multi-objective Genetic Algorithm (GA) that provides a power-performance Pareto-front under an accuracy constraint by selective multi-layer quantization during inference. Additionally, PiQi utilizes a neural network to expedite search time by predicting accuracy when assigning DNN layers to the appropriate cores.

CCS CONCEPTS

• **Computer systems organization** → **System on a chip**; **Embedded software**; • **Computing methodologies** → **Neural networks**; • **Software and its engineering** → *Software performance*; *Software usability*; **Embedded software**.

KEYWORDS

Edge Artificial Intelligence (Edge-AI), Low-Power Design (LPD), Partial Quantization, and Neural Processing Unit (NPU).

ACM Reference Format:
Ehsan Aghapour, Yixian Shen, Dolly Sapra, Andy D. Pimentel, and Anuj Pathania, University of Amsterdam, e.aghapour@uva.nl, y.shen@uva.nl, d.sapra@uva.nl, a.d.pimentel@uva.nl, a.pathania@uva.nl . 2024. PiQi: Partially Quantized DNN Inference on HMPSoCs. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '24), August 5–7, 2024, Newport Beach, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665314.3670841

ISLPED '24, August 5–7, 2024, Newport Beach, CA, USA

ACM ISBN 979-8-4007-0688-2/24/08.
https://doi.org/10.1145/3665314.3670841

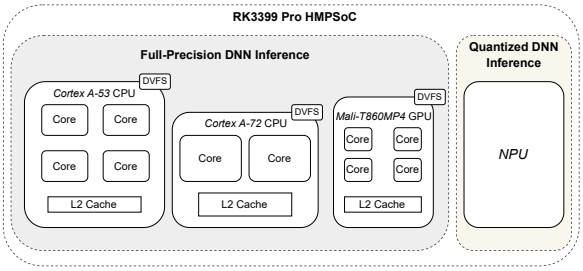


Figure 1: Abstract block diagram of the *RK3399Pro* HMPSoC.

1 INTRODUCTION

Deep Neural Networks (DNNs) are now commonplace in computer vision applications in embedded edge devices [7]. Heterogeneous Multi-Processor System-on-Chip (HMPSoC) platforms powering these edge devices allow for local on-chip DNN inference (without cloud support) for improved performance and privacy [6]. HMPSoCs ship with multiple inference-capable components such as Central Processing Units (CPUs) and Graphic Processing Units (GPUs) [10]. Moreover, state-of-the-art HMPSoCs are increasingly shipping with Neural Processing Units (NPUs) for DNN inference, as shown in Figure 1 for the *RK399Pro* HMPSoC.

NPUs are Application Specific Integrated Circuits (ASICs) that allow for low-latency, low-power on-chip inference [12]. Figure 2 shows the *NPU* provides a several-fold increase in inference power efficiency over a quad-core *Cortex-A53 CPU*, dual-core *Cortex-A72*, and quad-core *Mali-T860MP4* GPU running at peak frequency within an *RK399Pro* HMPSoC. However, the *NPU*, similar to most other NPUs in the market, can perform Int-8 quantized inference but at the cost of an accuracy loss from quantization. On the other hand, a *Cortex-A53 Little* CPU, *Cortex-A72 Big* CPU, and *Mali-T860MP4* GPU can perform FP-32 full-precision inference but with a magnitude lower power efficiency than the NPU. CPUs and GPUs can also perform quantized inference, but it only brings an accuracy loss with minimal gains in power efficiency relative to quantization with an NPU [11]. Figure 3 shows the Top-1 inference accuracy loss from Int-8 quantized inference against full-precision inference for different DNNs. The figure shows *AlexNet* and *GoogleNet* experience a minimal drop in accuracy due to quantization, while *YoLov3* and *MobileNet* experience a significant accuracy drop.

DNNs contain several heterogeneous layers [20]. In default inference implementations, all DNN layers execute with full precision on the CPU and GPU or with quantization on the NPU [21]. Consequently, if the accuracy requirement imposed by the user is higher than the accuracy with the NPU, then the system must employ inefficient CPU- or GPU-only inference to meet the accuracy constraint. Therefore, we present *PiQi*, a framework enabling the partial quantized inference on HMPSoCs for the first time. *PiQi* allows some DNN layer execution with full precision on CPU or GPU, whereas others execute quantized on the NPU. *PiQi* implements

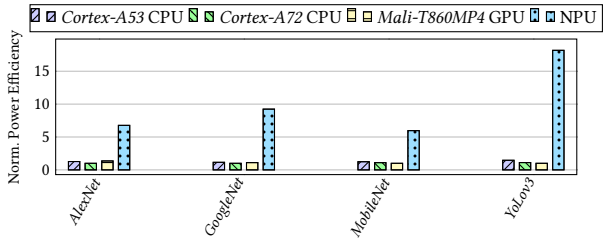


Figure 2: Power efficiency of different HMPSoC components with different DNNs on *RK3399Pro* HMPSoC.

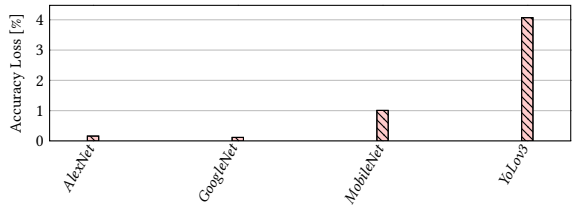


Figure 3: Top-1 Accuracy loss for different DNNs with Int-8 quantized inference over FP-32 full-precision inference.

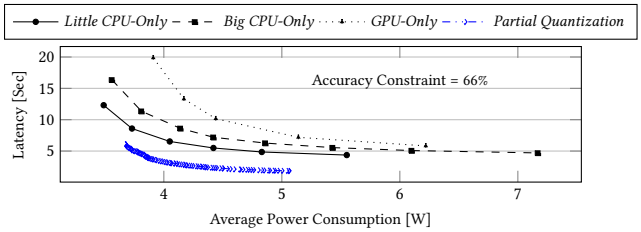


Figure 4: Power-Performance Pareto-front under different executions for *YoLov3* DNN under an accuracy constraint.

low-overhead mid-inference layer-level switching between CPU, GPU, and NPU to support partially quantized DNN inference.

Motivational Example: Figure 4 shows the Power-Performance Pareto-optimal front for the *YoLov3* DNN under a 66% accuracy constraint for various executions. *An accuracy constraint refers to the minimum level of user-defined accuracy that the final inference must achieve to meet operational requirements.* The accuracy (mAP) range for this model is 64.7% (fully quantized) to 68.7% (full precision). For the motivational example, we select a midrange target accuracy (66%) to explore the potential trade-offs between accuracy, power consumption, and latency. Since the NPU achieves only 64.7% with fully quantized inference, NPU-only quantized DNN inference is not feasible under the 66% constraint.

Figure 4 shows that even though full-precision CPU- or GPU-only DNN inference satisfy the accuracy constraint, the corresponding power-performance Pareto-fronts provide sub-optimal trade-offs between power and performance. The sub-optimality remains significant even with power-performance trade-offs enabled by CPU and GPU Dynamic Voltage Frequency Scaling (DVFS) in CPU- and GPU-only DNN inference, respectively. Figure 4 shows the near-optimal power-performance Pareto-front with *PiQi* using partially quantized DNN inference. Figure 4 shows that by synchronized use of CPU, GPU, and NPU during inference, along with CPU and GPU DVFS, *PiQi* provides a significantly better power-performance trade-off than any single-component DNN inference.

While reducing power consumption and execution time leads to lower energy consumption, our decision to optimize both power and time objectives reflects a holistic approach that captures a broader optimization landscape. By presenting the Pareto front of power and performance, we offer users diverse design options, each representing an optimal trade-off between power consumption and execution time. This approach addresses energy concerns and provides nuanced insights, empowering users to select the design point that best aligns with their specific constraints.

Novel Contributions: We make the following novel contributions in the scope of this work.

- We introduce the *PiQi* framework that enables partially quantized DNN inference on HMPSoCs by implementing layer-level switching between CPU, GPU, and NPU.
- We characterize the power, performance, and accuracy behaviour of DNNs under partial quantization and then build models to predict inference power, performance, and accuracy under multi-layer partial quantization.
- We provide a multi-objective Genetic Algorithm (GA) for determining a Power-Performance Pareto-front with partial quantization under an accuracy constraint.

Open-Source Contributions: The code for the *PiQi* is publicly available at *https://github.com/Ehsan-aghapour/PiQi* under *MIT* license. It utilizes the *ARM-CO-UP* framework described in [3].

2 RELATED WORK

Quantization [15] is a pivotal technique in DNN inference for enhancing latency and energy efficiency on edge devices in the literature. Notable studies by Nagel et al. [14] and Li et al. [13] have honed model accuracy through per-layer quantized weight optimization. Tann et al. [17] and Wang et al. [19] have furthered this by incorporating quantization noise into DNN training to fine-tune weights via Stochastic Gradient Design (SGD), minimizing accuracy loss. Jain et al. [8] addressed quantization-induced errors through dynamic compensation, albeit adding extra fix-point representation complexity. Ahn et al. [4] provided a performance characterization for quantization on edge devices, employing FP16/INT8 schemes on ARM CPU in *Raspberry Pi*. Coello et al. [5] developed a methodology for heterogeneously quantized DNN models, targeting minimum energy consumption and high accuracy with low latency.

In a closely related work, Tsuji et al. [18] presented a greedy search algorithm for solving the computationally hard combinatorial optimization problem of selective layer quantization under model-size constraints. A lightweight greedy algorithm works well under the time constraints imposed by live accuracy feedback. However, the greedy algorithm only explores a fraction of the exponential design space, missing out on better solutions.

3 IMPLEMENTATION

We implement partially quantized DNN inference using the *ARM Compute Library* (*ARM-CL*), which offers efficient low-level primitives for DNN execution on *ARM* CPUs and GPUs, particularly suited for *ARM*-based HMPSoCs such as *RK399Pro*[16]. Initially supporting only CPU- or GPU-only inference, *ARM-CL* has been extended with capabilities for mid-inference switching between CPU and GPU layers[1], along with DVFS enhancements for power

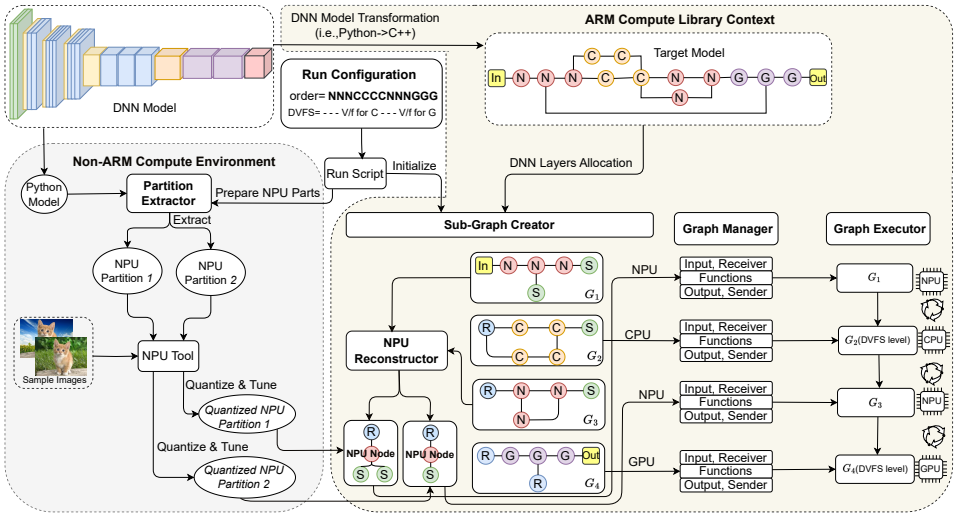


Figure 5: Illustrative abstraction for CPU, GPU, and NPU integration implementation

efficiency [2]. However, existing open-source frameworks do not integrate NPU support or allow seamless switching between CPU, GPU, and NPU mid-inference. For this purpose, we leverage our framework *ARM-COUP*, which integrates these features and is detailed in [3].

Figure 5 illustrates the *ARM-CL*-based implementation for the proposed *PiQi* framework. The implementation takes as inputs the DNN model and the desired `Run Configuration`, which specifies the partition of layers between CPU, GPU, and NPU. In the case of the *ARM big.Little* CPU, the CPU can be either a *big* or *Little* CPU for a given partition. It also describes the CPU and GPU layer-level DVFS settings. NPUs do not support DVFS.

The `Run Script` in the implementation then prepares the NPU partitions outside the *ARM-CL* context. It takes the *Python* model of the DNN and extracts out the parts marked for execution on NPU in the `Run Configuration`. It then uses the vendor-specific `NPU Tool`, in the case of this work from *Rockchip*, to quantize the NPU partitions. The `NPU Tool` tunes the partitions for minimal quantization-related accuracy loss using sample images from the training set. The partitions are now ready for use within *ARM-CL*.

The `Run Script` then invokes the *ARM-CL* context. *ARM-CL* converts the DNN model into an internal multi-node *C++* graph representation. *PiQi* uses a `Graph Creator` to break the *ARM-CL* graph into N sub-graphs for a `Run Configuration` with N component switches. Therefore, there is one sub-graph for each contiguous single-component execution. *PiQi* adds `Receiver` and `Sender` nodes to sub-graphs for the corresponding in and out connections. *PiQi* uses an `NPU Reconstructor` to convert all nodes in an NPU sub-graph, except the `Receiver` and `Sender` nodes, into a single NPU node. *PiQi* then replaces this NPU node with the corresponding quantized (and tuned) NPU partition prepared earlier. The NPU uses a `Graph Manager` to connect the sub-graphs. Finally, *PiQi* uses a `Graph Executor` to execute the connected sub-graphs. *PiQi* also applies layer-level CPU and GPU DVFS using the `Graph Executor`, as per the `Run Configuration`.

This implementation, which enables running the model on HMPSoC processors with a desired configuration, forms an essential operational component of *PiQi* . In contrast, the optimization process in PiQi, detailed in Section 5, utilizes the GA algorithm and prediction models for accuracy, power, and performance. It aims to identify the optimal configuration that achieves the target accuracy.

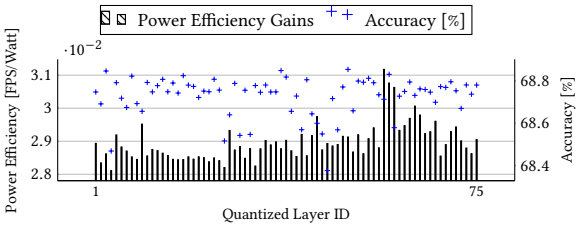
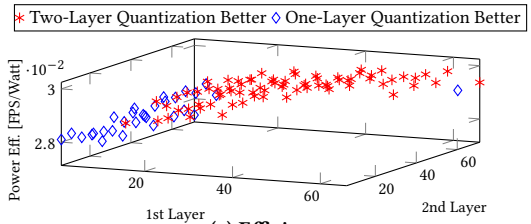


Figure 6: Power efficiency and accuracy for the full design space in one-layer quantization for *Yolov3*.

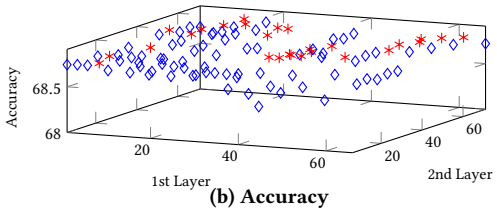
4 CHARACTERIZATION

Partially quantized inference with *PiQi* inherently involves executing some layers of the DNN on the NPU with quantization while executing other layers on a CPU or GPU with full precision. We perform a power-performance and accuracy loss characterization for partially quantized DNN inference. We use *YoLov3* as an example DNN for this characterization because of its large and complex neural architecture. Nevertheless, we make similar observations for *MobileNet*. There are $\binom{N}{X}$ design options for selecting X out of N DNN layers for quantization with *PiQi*. *YoLov3* comprises of 75 partitionable layers. Figure 6 shows the impact of executing a single layer (i.e., $\binom{75}{1}$ different design points) on the NPU with quantization and the remaining layers executing at full precision on the *Big* CPU running at full frequency. Figure 6 shows the entropy in power efficiency and accuracy with one-layer quantization.

These power, performance, and accuracy behaviours exhibit even higher entropy when multiple layers in a DNN are quantized together, as shown in Figure 7. Figure 7a and Figure 7b show the power efficiency and accuracy of $\binom{75}{2}$ design points (down-sampled by 5 for legibility) in two-layer quantization for *YoLov3*, respectively. Let tuple (A, B) represent the simultaneous quantization of layers A and B of the DNN. Intuition dictates power efficiency and



(a) Efficiency



(b) Accuracy

Figure 7: Power efficiency and accuracy for the down-sampled design space in two-layer quantization for *Yolov3*.

accuracy of two-layer quantization (A, B) versus either single-layer quantization (A) or (B) should be higher and lower, respectively.

Figure 7a compares the power efficiency of (A, B) with (A) and (B) for *YoLov3*. The figure shows the power efficiency of the two-layer quantization design point is better in most cases. In some cases where either of the one-layer quantization design points is better, we trace them to power-efficiency loss from the additional component switching overhead in the quantizing two non-contiguous layers dominating the power efficiency gains of additional quantization. Figure 7b compares the accuracy of (A, B) with (A) and (B) for *YoLov3*. The figure shows that two-layer quantization can lead to higher or lower accuracy loss. This observation stems from the fact that error propagation through the neural network architecture of a DNN is not well understood. It is possible for the errors from two quantized layers to cancel out partially.

5 *PIQI* FRAMEWORK

We introduce the *PiQi* framework for the partial quantization of DNNs on HMPSoCs. *PiQi* consists of modelling and optimization parts, as shown in Figure 8. The modelling part creates power, performance, and accuracy models for a user-specified DNN model. The optimization part finds the power-performance Pareto-front (using the models) for a user-defined accuracy constraint. *Using prediction models instead of live data allows several magnitudes faster evaluation of a partially quantized DNN configuration.*

Modelling. The modelling part of *PiQi* operates first by taking the user-specified DNN model as an input. A layer in the DNN can be either quantized or non-quantized. Therefore, 2^N possible partially quantized configurations (design points) are possible for a DNN with N layers. The design space is even larger as every non-quantized layer in the configuration can execute with full precision on a CPU or GPU at different DVFS levels. The quantization of a layer within the DNN often leads to the loss of certain feature information. This information loss propagates through the DNN, influencing subsequent computations to the last output layer and thus introducing errors in the prediction accuracy of the DNN. Determining the cumulative impact of information loss on prediction accuracy from multiple layers is non-trivial as the processed data undergoes multiple activation and soft-max functions before reaching the final output. Consequently, an analytical model for accuracy loss under partial quantization is hard to design. Therefore, we use a Machine Learning (ML) model to predict the accuracy.

Since training with the entire design space is not feasible, *PiQi* uses a sample generator to create a list of sample configurations across the design space. It uses all design points from 1- and 2-layer quantization, whereas it uses Monte Carlo sampling for 3-layer quantization and beyond. It then evaluates the accuracy of all of these configurations using a high-performance server. It takes the server 5 minutes on average to assess a configuration accuracy.

PiQi uses the accuracy data from sample configurations to train a dense neural network, as depicted in Figure 8, for predicting accuracy loss. The model comprises three fully connected layers with 128, 64, and 8 neurons, respectively, spanning from the input to the output layer. The input for the model is a binarized vector – 0/1 for non-/quantized-layer – the size of the number of layers N. *PiQi* tailors the accuracy predictor model for each DNN intended for deployment on the HMPSoC with an NPU using independent training. We train *Yolov3* and *MobileNet* models separately in this work with 18331 and 4689 data samples, respectively. It takes the server 5 minutes on average to train the model.

PiQi also predicts the power consumption and performance of a configuration. The authors of [2] provide an analytical power-performance model for mid-inference switching between CPU and GPU. The model also accounts for the overhead of component switching and CPU/GPU DVFS. We extend their model to account for overhead for back-and-forth switching with the NPU. The analytical model requires a power-performance profile for every layer on each component to work. We use an *Arduino*-based setup to directly profile layers on the CPU and GPU on the HMPSoC. However, in the case of the NPU, all layers are simultaneously loaded into an opaque vendor-specific NPU context. This context does not make it transparent when one layer ends and the other starts, making it infeasible for our profiling setup to attribute the power and performance to individual layers. Therefore, we use hybrid CPU-NPU configurations to profile an individual layer on the NPU while excluding the impact of data transfer and loading times. It takes the profiling setup 10 minutes on average to create a power-performance profile for a configuration. Most profiling time overhead originates from the one-time setup cost of loading the DNN model to the NPU and not from actual inference.

Optimization. The optimization part of *PiQi* follows the modelling part and takes in the user-defined accuracy constraint as input. *PiQi* uses a multi-objective Genetic Algorithm (GA) to determine the power-performance Pareto front under accuracy constraints for partially quantized DNN inference. *PiQi* performs a one-time encoding of configurations into chromosomes for the GA. The GA starts with randomly selected configurations as the initial population. It then uses the power, performance, and accuracy models to evaluate the fitness of configurations in the population. It takes less than a millisecond on average to evaluate the fitness of a configuration. A configuration in the population is unfit if it violates the accuracy constraint or is Pareto-dominated by another configuration in the population in terms of power and performance. The GA eliminates the unfit part of the population and uses mating functions (crossover and mutation) to produce new offspring from

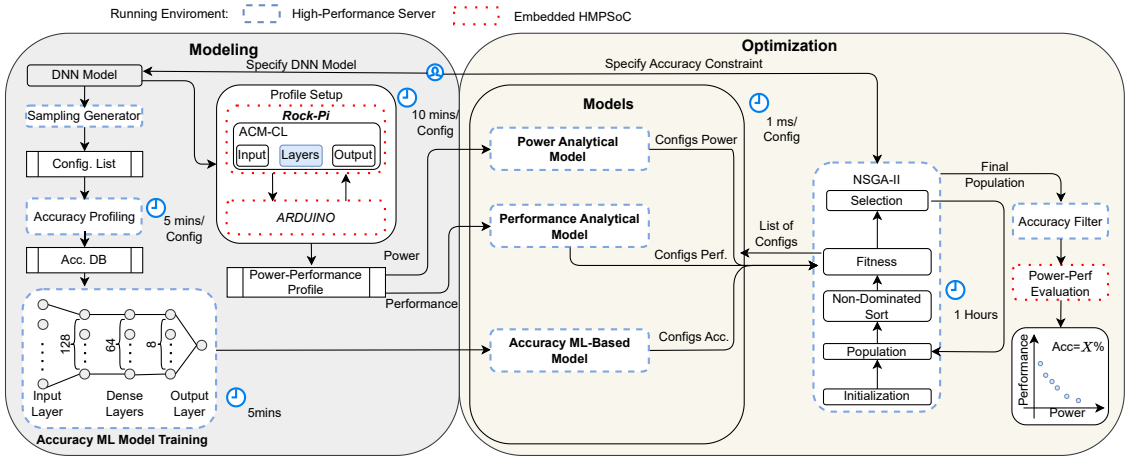


Figure 8: Abstract diagram showing the functioning for the proposed *PiQi* framework.

Table 1: The model-based prediction accuracy within *PiQi* .

Metric	*YoLov3*			*MobileNet*		
	MAE	Avg.	Samples	MAE	Avg.	Samples
Power	96 mW [1.7%]	5636 mW	300	226 mW [5.3%]	4217 mW	1000
Perf.	634 ms [9.8%]	6602 ms	300	9 ms [3.8%]	236 ms	1000
Acc.	0.0654%	68.0833%	2732	0.0258%	67.9117%	809

Table 2: *MobileNet* layer mappings for different accuracy.

Accuracy	Sample Layer Mapping	Num. Quantized Layers	Best Time [Overhead%]	Best Power
68.36	LLLLLLLLLLLLL	0	137 ms [0%]	3404 mW
68.36	GLLLLLNNNBNNN	6	100 ms [9%]	3229 mW
68.25	GLLLLNNNNNNNN	9	77 ms [10%]	3207 mW
68.15	GGNNNNNNNNNNN	12	63 ms [20%]	3158 mW
67.34	NNNNNNNNNNNNN	14	29 ms [0%]	3108 mW

the surviving configurations as replacements. The process iterates till the GA fails to produce stronger offspring for the population.

PiQi uses the *NSGA-II* algorithm [9] for the multi-objective GA. It runs on the server and converges to a solution on average in around one hour. Finally, the power-performance Pareto-front solution from the GA is adjusted based on real power, performance, and accuracy measurements to compensate for modelling errors before delivering the final results to the user.

6 EVALUATION

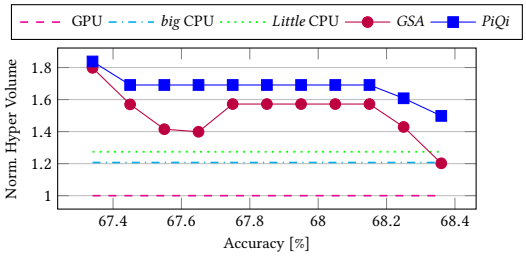
We evaluate *PiQi* using the *Rock-Pi N10* embedded platform containing an *RK3399Pro HMPSoC*, shown in Figure 1. Considering end-to-end latency and overall SoC power consumption, our reported results encompass all overheads, including processor switching, quantization and dequantization during NPU transitions, data transfer, and synchronization. We use two DNNs, *MobileNet* and *YoLov3*, that show significant enough accuracy loss with quantization (Figure 3) yet are also compatible with our extended *ARM-CL* implementation (Figure 5). *ARM-CL* integrates several models like *AlexNet* and *GoogleNet* in *Caffe* format incompatible with our partial quantization implementation. We use 50,000 and 5,000 images from the *ImageNet* and *COCO* datasets to evaluate the image classification and detection accuracy for *MobileNet* and *YoLov3*, respectively. We use *Keras* API for TensorFlow to train the accuracy models.

Prediction Results. Table 1 provides the prediction errors, measured as Mean Absolute Error (MAE), for power, performance, and accuracy models used within *PiQi*, demonstrating high modeling precision. The low error (MAE = 0.0654% for *YOLOv3* and 0.0258% for *MobileNet*) of the accuracy prediction model, can be attributed to the narrow variability in accuracy ranges observed across models (approximately 4% for *YoLov3 and 1% for* MobileNet).

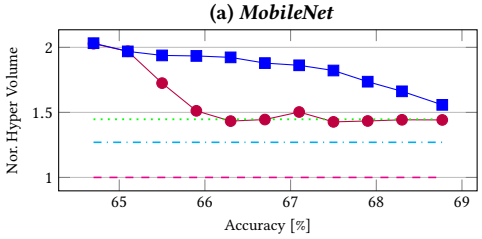
Baseline. No existing work proposes partially quantized DNN inference on HMPSoCs to the best of our knowledge. Therefore, we choose a recent greedy algorithm from [18] that solves a similar combinatorial optimization problem of selective layer quantization as *PiQi* as a baseline. The original greedy algorithm proposed in [18] is a single-objective optimization algorithm that provides a trade-off between accuracy and model size. We adapt the algorithm to select layers for quantization under an accuracy constraint and call the adaption Greedy Search Algorithm (*GSA*). *GSA* initializes by marking all layers for full-precision inference. It then greedily marks the layer expected to cause the minimum accuracy loss for quantization. It repeats the greedy marking process iteratively until further marking violates the accuracy constraint. The iteration ends with a valid design point wherein *GSA* marks each layer for full-precision or quantized execution. *GSA* then executes the design point on all full-precision components (CPUs or GPU) at all DVFS frequency levels to get the Power-Performance Pareto-frontier.

Optimization Results. Table 2 presents sample configurations detailing the layer mappings to NPU (N), GPU (G), *big* (B), and *Little* (L) CPU clusters alongside the maximum number of quantized layers for each target accuracy in *MobileNet*. It also showcases the latency and power consumption achieved among the Pareto-front results of *PiQi* for each target accuracy. Remarkably, even at the first target accuracy (68.36%), corresponding to the accuracy of the full precision model (first row), quantizing six selective layers (second row) leads to improvements in both time and power consumption. For a trade-off of only 0.21% accuracy (target accuracy=68.15), quantizing 12 out of 14 layers is feasible. Notably, while a fully quantized model would entail an accuracy drop to 67.34% (a 1.02% decrease, with the second layer exerting a relatively high impact on accuracy), selectively quantizing layers allows for enhanced power and latency performance with minimal accuracy compromise.

Overhead. The Best time column in table 2 includes the overhead shown in brackets, representing the percentage of time spent



(a) *MobileNet*



(b) *YoLov3*

Figure 9: Power-Performance Pareto-front hyper volume comparison between *GSA* and *PiQi* .

on switching processors and transferring data. The main overhead contributes to converting and loading to the NPU processor. As the output of the initial layers is larger, switching to NPU at earlier layers introduces more overheads. For example, one switch to the NPU from the second layer introduces high overhead for the 68.15% accuracy target case, while for the second row (target accuracy=68.36%), switching to the NPU happens at latter layers, which induces relatively less overhead (9% overhead for 4 switches).

Comparative Results. We evaluate *GSA* and *PiQi* for *YOLOv3* and *MobileNet* DNNs under different accuracy constraints within their achievable accuracy ranges. For each DNN, the accuracy constraints are bounded by the quantized-only and full-precision-only inference accuracy. Both *GSA* and *PiQi* use the same prediction model from Section 4 to ensure a fair comparison.

For each target accuracy (x-axis), we compare the multi-objective optimizations (power and latency Pareto-front) of the two methods using normalized hyper-volume as the metric. A fixed reference point is established for each DNN by selecting the maximum power consumption and latency among Pareto front design points of single components (refer to Figure 4). The reference points are (20320 ms, 7200 mW) for *YOLOv3* and (550 ms, 7000 mW) for *MobileNet*.

Figure 9 reports the model-based normalized hypervolume of the power-performance frontier obtained using the two methods for the selected accuracy constraints. While the Pareto frontier results of single-component inference remain fixed for different target accuracies, the power-latency Pareto frontier of *PiQi* consistently improves with decreasing target accuracy due to increased layer quantization. However, it's crucial to acknowledge that this improvement isn't consistent across all scenarios for *GSA*. In some intervals, particularly within *GSA*, where extensive switching occurs, there may be instances where the power-latency performance does not improve and may even degrade due to the overhead incurred from switching between processors and quantization/dequantization processes. The power-performance Pareto frontier provided by *PiQi* always dominates the frontier from *GSA*. Figure 9a and Figure 9b show *PiQi* providing, on average, 11.6% and 18.1% higher hypervolume than *GSA* (with the mentioned reference points) for *MobileNet* and *YoLov3*, respectively. The observed improvements are higher at accuracy constraints where the search space is larger.

7 CONCLUSION

We present the *PiQi* framework for partially quantized DNN Inference with NPUs found in state-of-the-art HMPSoCs. *PiQi* executes parts of the DNN inference with full precision on a CPU or GPU and other parts on the NPU with quantization. *PiQi* allows the use of the NPU in power-performance efficient DNN inference under accuracy constraints, wherein NPU-only execution is infeasible due to accuracy loss from quantization. *PiQi* has an open-source implementation for low-overhead switching between CPU, GPU, and NPU mid-inference to enable partially quantized DNN inference. It also has models for predicting power, performance, and accuracy loss for DNN inference with multi-layer partial quantization. Finally, it comes with a multi-objective GA that enables the determination of a power-performance Pareto-front under an accuracy constraint for partially quantized DNN inference. *PiQi* provides a significantly superior Pareto-front over the state-of-the-art.

REFERENCES

[1] Ehsan Aghapour et al. 2022. CPU-GPU Layer-Switched Low Latency CNN Inference. In *DSD*.
[2] Ehsan Aghapour et al. 2023. PELSI: Power-Efficient Layer-Switched Inference. In *RTCSA*.
[3] Ehsan Aghapour, Dolly Sapra, Andy Pimentel, and Anuj Pathania. 2024. ARM-CO-UP: ARM COoperative Utilization of Processors. *ACM Trans. Des. Autom. Electron. Syst.* (apr 2024). https://doi.org/10.1145/3656472 Just Accepted.
[4] Hyunho Ahn et al. 2023. Performance Characterization of using Quantization for DNN Inference on Edge Devices: Extended Version.
[5] Claudionor N. Coelho et al. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence* (2021).
[6] Xiaotian Guo et al. 2023. Automated Exploration and Implementation of Distributed CNN Inference at the Edge. *IoT Journal* (2023).
[7] Ramyad Hadidi et al. 2019. Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices. In *IISWC*.
[8] Shubham Jain et al. 2018. Compensated-DNN: Energy Efficient Low-Precision Deep Neural Networks by Compensating Quantization Errors. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*.
[9] Deb Kalyanmoy. 2002. A fast and elitist multi-objective genetic algorithm: NSGA-II. *TEVC* (2002).
[10] Andreas Karatzas et al. 2023. OmniBoost: Boosting Throughput of Heterogeneous Embedded Devices under Multi-DNN Workload. In *DAC*.
[11] Youngsok Kim et al. 2019. Layer: Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization. In *EuroSys*.
[12] Kyuho J. Lee. 2021. Architecture of neural processing unit for deep neural networks. In *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*. Elsevier.
[13] Yuhang Li et al. 2021. BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction. arXiv:2102.05426
[14] Markus Nagel et al. 2019. Data-Free Quantization Through Weight Equalization and Bias Correction. In *ICCV*.
[15] Markus Nagel et al. 2021. A White Paper on Neural Network Quantization. arXiv:2106.08295
[16] Jie Tang et al. 2017. Enabling Deep Learning on IoT Devices. *Computer* (2017).
[17] Hokchhay Tann et al. 2017. Hardware-Software Codesign of Accurate, Multiplier-Free Deep Neural Networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*.
[18] Satoki Tsuji et al. 2022. Greedy search algorithm for partial quantization of convolutional neural networks inspired by submodular optimization. *Neural Computing and Applications* (2022).
[19] Peisong Wang et al. 2023. Optimization-Based Post-Training Quantization With Bit-Split and Stitching. *TPAMI* (2023).
[20] Siqi Wang et al. 2020. High-Throughput CNN Inference on Embedded ARM Big.LITTLE Multicore Processors. *TCAD* (2020).
[21] Siqi Wang et al. 2020. Neural Network Inference on Mobile SoCs. *IEEE Design Test* (2020).