RankMap: Priority-Aware Multi-DNN Manager for Heterogeneous Embedded Devices

Andreas Karatzas[1], Dimitrios Stamoulis[2], Iraklis Anagnostopoulos[1]
[1]School of Electrical, Computer, and Biomedical Engineering, Southern Illinois University, Carbondale, IL, U.S.A.
[2]Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, U.S.A.
Email: {andreas.karatzas, iraklis.anagno}@siu.edu, dstamoulis@utexas.edu

***Abstract*—Modern edge data centers simultaneously handle multiple Deep Neural Networks (DNNs), leading to significant challenges in workload management. Thus, current management systems must leverage the architectural heterogeneity of new embedded systems to efficiently handle multi-DNN workloads. This paper introduces RankMap, a priority-aware manager specifically designed for multi-DNN tasks on heterogeneous embedded devices. RankMap addresses the extensive solution space of multi-DNN mapping through stochastic space exploration combined with a performance estimator. Experimental results show that RankMap achieves $\times 3.6$ higher average throughput compared to existing methods, while preventing DNN starvation under heavy workloads and improving the prioritization of specified DNNs by $\times 57.5$.**

***Index Terms*—Deep Neural Networks, Multi-DNN Workloads, Heterogeneous Architectures, Edge Inference, DNN Performance Prediction, Prioritization, Application Starvation**

I. Introduction

We are in an era of rapid machine learning advancements, where Deep Neural Networks (DNNs) play an important role in modern embedded systems. Embedded systems often utilize heterogeneous computing components to manage the computational demands of DNNs. However, run-time managers neglect the underlying heterogeneity [1]. Current deep learning frameworks typically utilize either the CPU or the GPU exclusively, thus underutilizing the system's diverse components [2].

Additionally, modern embedded devices must address the challenge of deploying multi-DNN workloads, where multiple DNNs run simultaneously on a single system. Efficiently managing the available computing resources in these scenarios becomes a critical issue. A way to solve this problem is to map each DNN on one of the available processing units [3]. However, these coarse-grained methods are far from optimal [4].

Moreover, the prioritization of each DNN within a multi-DNN workload, an often overlooked aspect, adds significant complexity to the problem. Meeting Service Level Agreement (SLA) requirements become particularly challenging, making prioritization a crucial metric for evaluating a multi-DNN manager [5]. Users are categorized into different SLA groups, leading to multi-DNN workloads where each DNN has a different priority level. For instance, in edge data centers where multiple users submit DNN queries, *current runtime managers lead to reduced throughput in* 91% *of cases and starving some DNNs* 30% *of the time, resulting in an overall unsatisfactory user experience* (see Section II).

In this work, we present **RankMap**, a framework for efficient multi-DNN management onto heterogeneous embedded devices. RankMap learns the computational profile of the individual DNN layers and groups them into pipeline stages to map them among the given computing components. It reduces resource contention and boosts system throughput. RankMap also ensures that each DNN receives enough computing resources proportional to its priority (rank) without starving other DNNs running concurrently. RankMap outperforms previous approaches, achieving up to $\times 3.6$ higher average throughput across all multi-DNN scenarios, while satisfying priority constraints $\times 57.5$ more efficiently, ensuring no DNN is starved.

Overall, our main contributions are: ❶ We propose RankMap, a lightweight and scalable multi-DNN manager that utilizes fine-grained DNN segmentation to boost system throughput on heterogeneous embedded devices. ❷ RankMap creates mappings for multi-DNN workloads considering each DNN's priority to satisfy their performance requirements, avoiding starvation effects. ❸ RankMap employs a novel multi-task attention-based CNN for throughput estimation of any multi-DNN scenario. A Monte Carlo Tree Search (MCTS) algorithm utilizes this estimator as a feedback mechanism to efficiently explore the vast solution space of those mappings.

II. Motivation

In this section, we highlight the importance of an efficient multi-DNN manager. Specifically, we demonstrate ❶ the impact of DNN partitioning on system throughput, and ❷ potential starvation issues in multi-DNN workloads. We target scenarios where each DNN is an independent task, meaning the output of one DNN does not serve as input for another. This setup is particularly relevant for edge data centers serving multiple users, each submitting different DNN requests. We focus on Computer vision applications in our experiments; even with the rise of Large Language Models (LLMs) and Generative AI, vision models: (**i**) remain more prevalent in edge environments over natural language ones [6], (**ii**) are typically executed on edge hardware more efficiently [7], as (**iii**) major hardware providers like ARM and NVIDIA offer more robust support for computer vision at the edge compared to natural language tasks [8], [9]. To that end, we select four diverse and widely used DNNs for our multi-DNN workload: SqueezeNet-V2, Inception-V4, ResNet-50, and VGG-16. We utilize the Orange Pi 5 board, which features a Mali-G610 GPU and a big.LITTLE CPU, consisting of a quad-core Cortex-A76 at 2.4GHz and a quad-core Cortex-A55 at 1.8GHz. We introduce the potential throughput metric $\mathcal{P}$ to quantify a

978-3-9826741-0-0/DATE25/© 2025 EDAA

DNN's performance in a multi-DNN environment relative to its isolated performance, *offering insights into resource utilization efficiency*. For a DNN i, $\mathcal{P}^i = \frac{t^i_{current}}{t^i_{ideal}}$, where $t^i_{current}$ is the throughput of DNN i (in inferences per second) when running alongside other DNNs, and t^i_{ideal} is its throughput when running alone on the GPU.

First, we mapped all the DNNs onto the GPU, our baseline configuration, as it's traditionally preferred over the CPU for its superior computing capabilities. Next, we generated 300 unique random mappings of the selected multi-DNN workload. In these mappings, each DNN was split into arbitrary groups of contiguous layers, forming pipeline stages. These stages were then randomly assigned to the three available computing components—the big CPU cluster, the LITTLE CPU cluster, and the GPU—resulting in a distributed pipeline execution across the heterogeneous system. Figure 1 presents the throughput distribution normalized to the baseline. For each mapping, throughput is defined as $\mathcal{T} = \frac{\sum_{i=1}^{N} t^i_{current}}{N}$, where N is the number of DNNs in the workload—four in our case. Figure 2 shows the distribution of the potential throughput $\mathcal{P}$ for each DNN. From these figures, we observe the following:

Key Observations

❶ The baseline heavily saturates the GPU, and hence 91% of mappings exhibit better average throughput $\mathcal{T}$. This proves the efficiency of partitioning DNNs across all the available computing components.

❷ A multi-DNN manager is likely to suboptimally allocate the DNNs across the computing components. This can significantly reduce the performance of at least one DNN and starve it. Specifically, in Figure 1, we observe that the risk of starvation is almost a certainty when we go beyond the threshold of $\mathcal{T} \geq 2.4$. Mappings with the highest throughput $\mathcal{T}$ often come at the expense of other DNNs in the workload, with 30.2% of mappings causing starvation.

❸ Deeper and more complex architectures, such as Inception-V4 and VGG-16, are very challenging due to their high probability of starvation. Specifically, in Figure 2, we observe that the mean $\mathcal{P}$ for Inception-V4 is around 0.1.

❹ Finding mappings that consistently satisfy minimum performance criteria is improbable. In Figure 2, we observe that going beyond the threshold of $\mathcal{P} \geq 0.6$ means some DNNs will underperform.

❺ In Figure 2, we observe that more than 60% of DNNs will exhibit $\mathcal{P} \leq 0.2$. Thereby, to go beyond that barrier, we have to research AI-powered multi-DNN managers that efficiently correlate the hardware properties to the queried multi-DNN workloads.

Overall, supporting priorities for DNNs to *ensure they meet performance goals while maintaining high system throughput and preventing application starvation* is a challenging and unexplored problem.

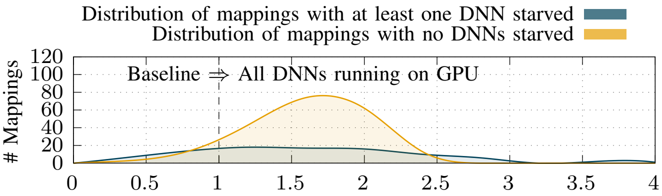


Fig. 1: Average throughput $\mathcal{T}$ normalized by baseline.

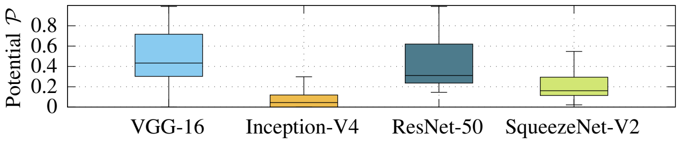


Fig. 2: Potential Throughput $\mathcal{P} \forall$ DNNs.

III. Related Work

The authors in [2] propose SURF, which employs heuristic techniques for DNN partitioning to construct efficient DNN pipelines. Similarly, Pipe-it [10] is a framework that predicts the performance of different layers to create DNN pipelines. However, it targets the CPU only. Furthermore, the authors in [11] propose the ARM-CO-UP framework, which boosts throughput by employing pipelined execution of DNN partitions. However, they do not consider concurrent execution of different DNNs. BAND [12] focuses on identifying and grouping DNN subgraphs that utilize similar computing operations. Moreover, the authors in [13] present a method that allocates different layers to the system's various processing units to prioritize immediate resource assignment. However, they follow a greedy approach, which is not scalable. MOHaM [14] co-optimizes hardware mapping for multi-DNN workloads. However, it designs sub-accelerators rather than focusing on DNN partitioning or concurrent DNN execution on shared resources. Similarly, the authors in [15] present a heuristic method to support multi-DNN workloads. However, they employ multiple accelerators operating on different precisions that may not be available on conventional embedded devices. Furthermore, the authors in [16] demonstrate a method to pipeline the parallelism of DNNs by partitioning them into smaller stages, thereby reducing inference latency. However, their method targets larger systems and does not support priorities. Likewise, MOSAIC [17] uses DNN partitioning for workload distribution, relying on a linear regression model that correlates layer input sizes with computational needs, trained on single DNN cases. This method overloads the embedded GPU and cannot support DNNs with different priorities. ODMDEF [18] creates pipelines for handling multiple DNN workloads. This method, though, needs a considerable amount of data to achieve reliable accuracy and does not support priorities either. The authors in [3] introduce a framework based on an evolutionary algorithm to partition the DNNs on the given computing components. However, this method converges slowly and does not scale well due to the constant re-training required to support different scenarios. Additionally, the authors in [4] present OmniBoost, a framework that uses a CNN to estimate the performance of DNN layers on the computing components of the embedded device. Even though OmniBoost achieves high average throughput, it does not support priorities and requires extensive profiling of all the

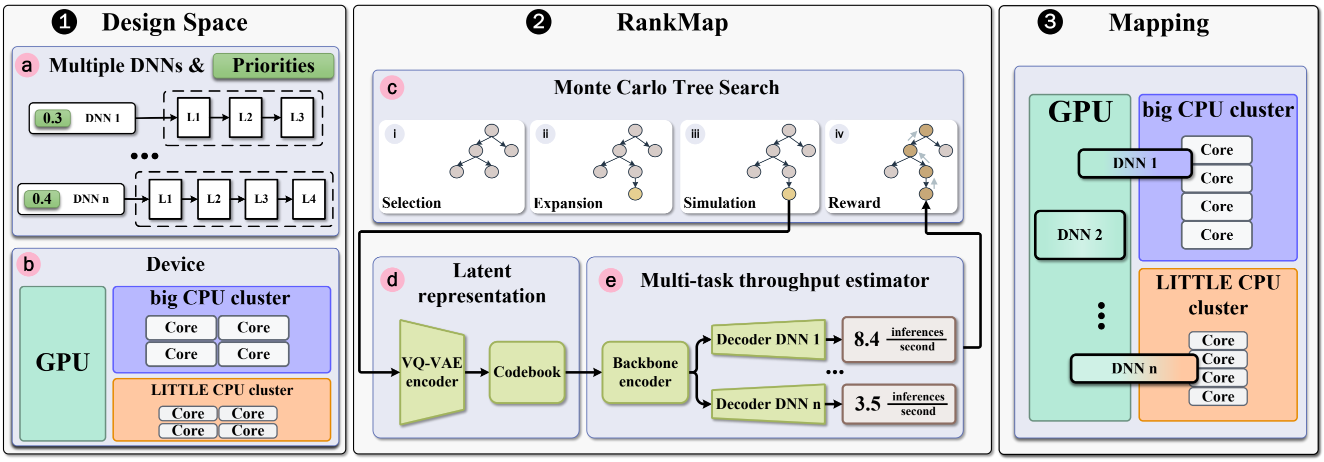


Fig. 3: A high-level overview of RankMap.

layers of a DNN. Our key differences from the state-of-the-art are manifold: (**i**) We efficiently explore the design space by employing MCTS for pruning a decision tree based on highly accurate feedback from our throughput estimator; (**ii**) RankMap accounts for each DNN's priority to generate a mapping; and (**iii**) We prevent DNN starvation regardless of the workload.

IV. Proposed Framework

This section introduces RankMap, a multi-DNN manager for heterogeneous embedded systems. Figure 3 depicts the overview of the proposed approach. RankMap takes three key inputs: (**i**) A set of DNNs to be executed concurrently; (**ii**) A collection of available computing components, such as big CPU cluster, LITTLE CPU cluster, and GPU; and (**iii**) A list of application ranks indicating the priority of each DNN in the workload. RankMap addresses the following objectives: ❶ Maintain high system throughput; ❷ Account for individual DNN priorities; ❸ Prevent application starvation.

To this end, RankMap employs a fine-grained approach to workload distribution: (**i**) **DNN partitioning:** Each DNN is divided into smaller sub-DNNs; (**ii**) **Adaptive Mapping:** These sub-DNNs are strategically allocated across the system's diverse computing components; and (**iii**) **Performance Optimization:** This granular distribution enhances the performance of high-priority DNNs, boosts overall system throughput, and accounts for resource allocation requirements for all DNNs.

A. Layer representation

We first define the input tensor $\mathcal{Q}$ to represent a multi-DNN mapping. Each channel $\mathbf{c}^i$ in $\mathcal{Q}$ corresponds to the i^{th} DNN in the workload. Each row j in $\mathbf{c}^i$ represents a layer $\mathbf{l}^i_j$. We divide each channel $\mathbf{c}^i$ into $\mathbf{d}$ column blocks, where each block represents a different computing component. To capture the complexity and size of each layer, we use the 22-dimensional vector formulated in Equation 1.

$$\mathbf{l}^i_j = \Big(\underset{\text{Layer index of DNN } i}{j} \quad \overset{\text{Layer type}}{\mathbf{t}} \quad \underset{\text{Input feature map}}{\mathbf{ifm}} \quad \overset{\text{Output feature map}}{\mathbf{ofm}} \quad \underset{\text{Weights tensor}}{\mathbf{w}} \quad \overset{\text{Number of biases}}{\mathbf{b}} \quad \underset{\text{Type of activation}}{\mathbf{a}} \quad \overset{\text{Pad-stride information}}{\mathbf{ps}} \Big) \tag{1}$$

Tensors **ifm**, **ofm**, and **w** comprise 4 elements: (**i**) the minibatch size; (**ii**) the number of channels; (**iii**) the height of the feature map; and (**iv**) the width of the feature map. Finally, **ps** is a 6-dimensional tensor representing the layer's pad-stride information.

B. Static and Dynamic Prioritization

We employ two approaches within the RankMap framework to prioritize DNNs, effectively addressing different operational needs and scenarios. The static priority method is designed for scenarios where a specific DNN is prioritized above others within the workload. This DNN is assigned a high priority $\mathbf{p}^i$. This ensures that resources are allocated preferentially to this DNN to meet specific performance requirements.

In contrast to static priority, dynamic priority adjusts the importance of each DNN based on its computational demands as characterized by the layer profile $\mathbf{l}^i_j$. This method uses a priority vector that changes dynamically, facilitating more balanced resource distribution across all DNNs in the workload. This flexibility results in higher overall system throughput compared to the static method (Section V), as it allows the system to adapt resource allocation based on real-time computational needs. Both priority types are designed to prevent application starvation. To ensure no DNN is starved, RankMap employs a disqualification mechanism, as detailed in Section IV-E. Any mapping likely to result in starvation is automatically excluded, thus safeguarding against performance degradation and ensuring efficient resource distribution.

C. Vector Quantized-Variational AutoEncoder

Next, we utilize a Vector Quantized Variational AutoEncoder (VQ-VAE) model [19] to effectively compress and encode the raw layer vector representations. The VQ-VAE includes a bottleneck layer that, unlike typical AutoEncoders, introduces a controlled level of stochasticity. It quantizes the data distribution using discrete latent variables, enabling the creation of a highly scalable codebook. Each layer is compressed into a 16-dimensional distributed embedding [20], reducing the computational load of our throughput estimator by $\sim 58\%$ in multiply-accumulate (MAC) operations. To convert the raw layer vectors into latent vectors, we utilize 1D convolutional layers. We apply quantization to the distribution in the latent space using Grouped Residual Vector Quantization [21].

D. Throughput Estimator

Building on the formulation of the input tensor $\mathcal{Q}$, we evaluate any mapping $\mathcal{M}$ using a specialized, multi-task, attention-based convolutional neural network (CNN). This lightweight CNN has approximately $3,7$M parameters and is designed to predict the throughput of each DNN in the workload, expressed in inferences per second, for any given mapping. We approach the throughput estimation for each DNN as an individual task. The multi-task nature of our model allows it to focus on the unique characteristics of each network.

The performance estimator incorporates three shared residual layers that build the backbone and one decoder stream per DNN in the multi-DNN workload. The residual layers are a stack of: **(i)** $\times 2$ depth-wise 2D convolutional layers and self-attention [22] modules; and **(ii)** a 2D convolutional layer followed by batch normalization. Contrary to standard 2D convolutions where channels usually exhibit spatial correlations, the channels in $\mathcal{Q}$ are statistically independent. This necessitates depth-wise convolutions and self-attention modules. Finally, the decoder streams comprise a linear attention layer [23], followed by two fully connected layers. Since our performance estimator solves a multi-task problem, linear attention efficiently attends to the feature subset of the encoder concerning each task.

E. Monte-Carlo Tree Search

While the throughput estimator predicts a DNN's performance, finding the optimal mapping on the platform is *challenging due to the vast solution space*. Consider the following multi-DNN workload as an example: **(i)** AlexNet; **(ii)** MobileNet; **(iii)** ResNet-50; and **(iv)** ShuffleNet. For this combination alone, the total number of possible mappings is calculated as $3^{(8+20+18+18)} \approx 4\text{E}{+}10$. Here, the numbers $\{8, 20, 18, 18\}$ represent the valid partition points for each DNN, and 3 corresponds to the number of computing components (i.e., GPU, big CPU cluster, and LITTLE CPU cluster). Evaluating such a vast number of mappings is impractical.

To address this, we use Monte-Carlo Tree Search (MCTS) [24], which stochastically prunes the search space through a decision tree. MCTS iteratively selects and expands tree nodes based on their rewards, with each node representing a potential mapping. During expansion, a random path under a selected node is explored, using the trained performance estimator to simulate these paths. The estimator's output updates the decision tree's weights, which, combined with a predefined computational budget, prunes the search tree for fast and accurate results. The final output is the mapping with the highest expected reward.

To factor in DNN priorities, we use the priority vector $\mathbf{p}$ to calculate the reward for a given mapping $\mathcal{M}$. The throughput estimator's output $\mathbf{O}(\mathcal{M})$ is multiplied by $\mathbf{p}$ to generate the reward. *If any element in* $\mathbf{O}(\mathcal{M})$ *falls below a specific threshold* $\mathbf{th}$, *the state is disqualified from the solution space*. Overall, MCTS solves for the optimal mapping $\mathcal{M}^*$:

$$\mathcal{M}^* = \arg\max_{\mathcal{M}} \left(\mathbf{O}(\mathcal{M})^{\mathbf{T}} \times \mathbf{p} \;\middle|\; \mathbf{O}(\mathcal{M})^{\mathbf{i}} > \mathbf{th} \;\; \forall \mathbf{i} \in \mathbf{O}(\mathcal{M}) \right)$$

which favors mappings meeting $\mathbf{th}$, with their reward reflecting the weighted priorities of the DNNs.

V. Experimental Evaluation

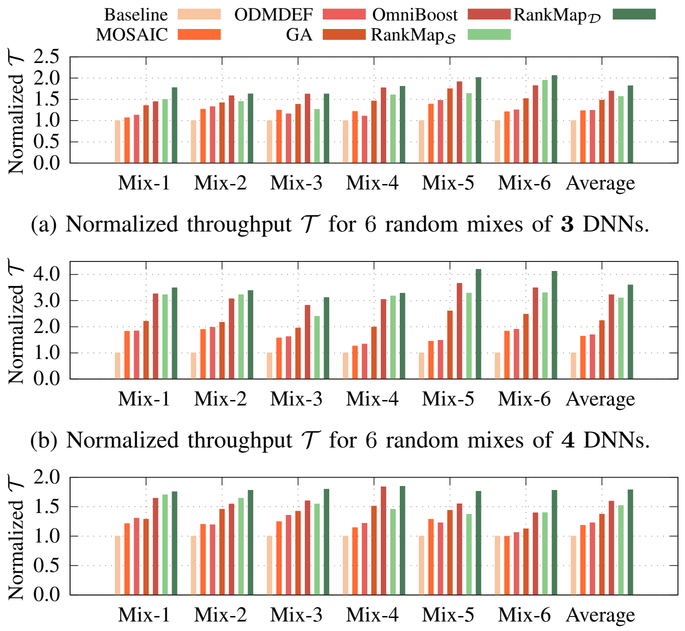
In this section, we demonstrate the efficiency of RankMap in terms of: **(i)** increased average system throughput and enhanced performance for high-priority DNNs (Section V-A); **(ii)** prevention of starvation (Section V-B); **(iii)** correlation between throughput and priorities (Section V-C); and **(iv)** run-time trade-off (Section V-D). We create several diverse multi-DNN workloads on the Orange Pi 5 board, a heterogeneous embedded platform featuring a Mali-G610 GPU and big.LITTLE CPUs with quad-core Cortex-A76 and quad-core Cortex-A55 running at 2.4GHz and 1.8GHz, respectively. RankMap is implemented in PyTorch. ARM Compute Library [25] and OpenCL are used to execute DNNs on the board and for DNN partitioning.

To train our multi-task throughput estimator, we created a dataset of 10K workloads. Each workload consists of a mix of up to 5 concurrent DNNs randomly selected from a pool of 23 DNNs. The DNNs in the pool are: **(i)** AlexNet, **(ii)** DenseNet-121, **(iii)** DenseNet-169, **(iv)** EfficientNet-B0, **(v)** EfficientNet-B1, **(vi)** EfficientNet-B2, **(vii)** GoogleNet, **(viii)** Inception-ResNet V2, **(ix)** Inception V3, **(x)** Inception V4, **(xi)** MobileNet, **(xii)** MobileNet V2, **(xiii)** ResNet-12, **(xiv)** ResNet-50, **(xv)** ResNet-50 V2, **(xvi)** ResNeXt-50, **(xvii)** ShuffleNet, **(xviii)** SqueezeNet, **(xix)** SqueezeNet V2, **(xx)** SSD with MobileNet backbone, **(xxi)** YOLO V3, **(xxii)** VGG-16, and **(xxiii)** VGG-19.

We randomly partitioned each DNN and mapped the sub-DNNs across the device's computing components, creating a diverse and sizable dataset. This randomness ensures the dataset represents the entire solution space, with each sample being unique. We executed each workload on the board, recording the throughput for each DNN. We collected 10K samples, using 90% for training our throughput estimator. Although the estimator will be tested on real-world unseen scenarios during experiments, we reserved 10% of the dataset for feedback during training. Using L2-loss for each decoder stream, our estimator achieved an L2 loss of about 0.14 after 50 epochs. Random channel shuffling as a dataset augmentation step further reduced the L2 loss to about 0.08.

We consider the following metrics: **(i)** **Normalized throughput** $\mathcal{T}$: The system throughput is calculated as $\mathcal{T} = \frac{\sum_{i=1}^{N} t^i_{current}}{N}$, where $t_{current}$ represents the throughput in inferences per second for each DNN i when operating concurrently within the workload. This total throughput is then normalized against the baseline, i.e., the scenario where all DNNs are executed on the GPU, the highest-performing component on the Orange Pi 5; and **(ii)** **Potential throughput** $\mathcal{P}$: Defined as $\mathcal{P} = \frac{t^i_{current}}{t^i_{ideal}}$, where t_{ideal} is the throughput of the DNN when executed alone on the GPU. This metric assesses how the performance of each DNN in a shared environment compares to its performance in isolation. Potential throughput provides a quantitative measure to see how close a DNN's current performance is relative to its optimal performance.

Additionally, in the following experiments, we evaluated RankMap against: **(i)** **Baseline**; **(ii)** **MOSAIC** [17], a linear



(a) Normalized throughput $\mathcal{T}$ for 6 random mixes of **3** DNNs.

(b) Normalized throughput $\mathcal{T}$ for 6 random mixes of **4** DNNs.

(c) Normalized throughput $\mathcal{T}$ for 6 random mixes of **5** DNNs.

Fig. 4: Normalized throughput $\mathcal{T}$ across random mixes of **3**, **4**, and **5** concurrent DNNs.

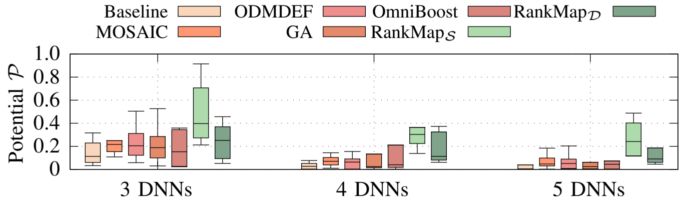


Fig. 5: Potential throughput $\mathcal{P}$ of the high-priority DNN across mixes of **3**, **4**, and **5** concurrent DNNs.

regression approach; **(iii)** **ODMDEF** [18], a manager with both a linear regression and a k-NN classifier in its core; **(iv)** **GA**, the evolutionary manager proposed in [3]; and **(v)** **OmniBoost** [4], a framework for greedy throughput optimization of multi-DNN workloads. Regarding RankMap, we considered: **(i)** **Static mode** RankMap$_\mathcal{S}$, where a high priority is assigned to a critical DNN operating concurrently with others; and **(ii)** **Dynamic mode** RankMap$_\mathcal{D}$, where the priority vector for each DNN is derived from its profiling stage, as detailed in Section IV-B.

A. *Throughput and Prioritization comparison*

To evaluate the efficiency of RankMap in terms of throughput and prioritization, we created multi-DNN workloads with 3, 4, and 5 DNNs operating concurrently. For each set of workloads, we evaluated both the normalized throughput $\mathcal{T}$ and the potential throughput $\mathcal{P}$. First, we focus on the potential throughput of the DNN with the *highest* priority to highlight how well each manager evaluates key applications and supports the critical DNN in a workload (extensive analysis of all DNNs' performance with respect to their assigned priority to illustrate RankMap's capabilities in managing diverse DNN workloads follows in Section V-C).

Mixes of 3 DNNs: Figure 4a shows that RankMap$_\mathcal{D}$ achieves on average 82%, 48%, 46%, 23%, and 8% higher $\mathcal{T}$ than the Baseline, MOSAIC, ODMDEF, GA, and OmniBoost, respectively. The RankMap$_\mathcal{S}$ falls behind the dynamic by 15%. However, RankMap$_\mathcal{S}$ keeps the potential throughput $\mathcal{P}$ higher by ×6.3, ×2.2, ×2.7, ×4, and ×2.6 than the Baseline, MOSAIC, ODMDEF, GA, OmniBoost, and RankMap$_\mathcal{D}$, respectively (Figure 5). Notably, RankMap$_\mathcal{S}$ maintains higher $\mathcal{P}$ than 0.21 under any workload, with a peak value of 0.91.

Mixes of 4 DNNs: Regarding 4 concurrent DNNs, we observe that some managers have already overwhelmed the GPU. Specifically, Figure 4b depicts that RankMap$_\mathcal{D}$ demonstrated ×3.6, ×2.2, ×2.1, ×1.6, and ×1.2 higher normalized throughput $\mathcal{T}$ compared to the Baseline, MOSAIC, ODMDEF, GA, and OmniBoost, respectively. RankMap$_\mathcal{S}$ fell behind by 14% in terms of $\mathcal{T}$ compared to RankMap$_\mathcal{D}$. However, Figure 5 justifies this trade-off since RankMap$_\mathcal{S}$ demonstrated higher $\mathcal{P}$ by ×57.5, ×7.4, ×35.1, ×21.9, and ×2.2 than the Baseline, MOSAIC, ODMDEF, GA, OmniBoost, and RankMap$_\mathcal{D}$, respectively. Notably, RankMap$_\mathcal{S}$ keeps $\mathcal{P}$ higher than 0.14 under any workload, with a peak value of 0.37. Hence, under heavy workload, RankMap$_\mathcal{S}$ satisfies the critical DNN's performance requirements while maintaining high system throughput.

Mixes of 5 DNNs: To further stress the device's resources, we evaluate the managers for 5 concurrent DNNs (Figures 4 and 5). While under heavy workload, RankMap$_\mathcal{D}$ still finds solutions that increase the average system throughput by 79%, 51%, 46%, 30%, and 12% compared to the Baseline, MOSAIC, ODMDEF, GA, and OmniBoost, respectively. RankMap$_\mathcal{S}$ falls again third behind RankMap$_\mathcal{D}$ and OmniBoost by 5% and 17%, respectively. Still, it yields ×55, ×12, ×18, ×42, and ×38 higher $\mathcal{P}$ than the Baseline, MOSAIC, ODMDEF, GA, and OmniBoost, respectively. Furthermore, RankMap$_\mathcal{S}$ keeps $\mathcal{P}$ higher than 0.12 under any workload, with a peak value of 0.49. Hence, RankMap$_\mathcal{S}$ effectively prioritizes DNNs even under heavy workloads, while also boosting system throughput.

B. *Starvation comparison*

Next, we determine if there were any instances of DNN starvation within the workload mixes discussed in Section V-A. To that end, we analyzed the occurrence of starved DNNs across the 72 samples from these mixes, as illustrated in Figure 6. We define a DNN as starved when its potential throughput ($\mathcal{P}$) is 0, represented by the red bin in Figure 6. Our findings show that both the RankMap$_\mathcal{S}$ and RankMap$_\mathcal{D}$ successfully prevent DNN starvation under all tested workloads. In contrast, Baseline, MOSAIC, ODMDEF, GA, and OmniBoost returned mappings that starved some DNNs, with respective counts of 19, 9, 13, 11, and 5 starved DNNs out of 72. Therefore, in pursuit of higher throughput, some managers caused starvation effects by throttling certain DNNs.

C. *Throughput and Priority correlation*

In this section, we evaluate how effectively RankMap satisfies the assigned priorities for all DNNs in the mix. As mentioned earlier, RankMap$_\mathcal{D}$ assigns priorities to the DNNs based on their computational profiles. Thus, for RankMap$_\mathcal{D}$, we

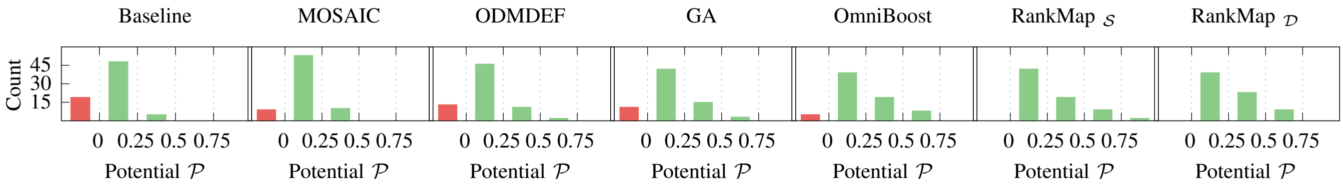
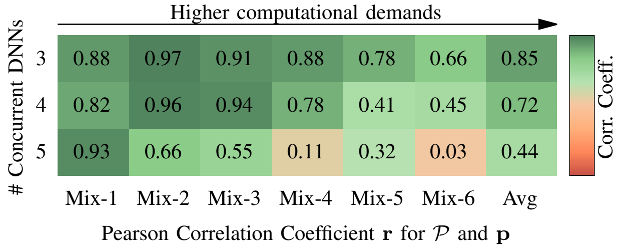


Fig. 6: Comparison of Potential Throughput $\mathcal{P}$ across all the performed experiments.



	Mix-1	Mix-2	Mix-3	Mix-4	Mix-5	Mix-6	Avg
3	0.88	0.97	0.91	0.88	0.78	0.66	0.85
4	0.82	0.96	0.94	0.78	0.41	0.45	0.72
5	0.93	0.66	0.55	0.11	0.32	0.03	0.44

Fig. 7: Pearson corr. coef. of Potential Throughput $\mathcal{P}$ and priority vectors $\mathbf{p}$ for the mixes of 3, 4, and 5 concurrent DNNs for RankMap$_\mathcal{D}$.

compiled a heatmap in Figure 7 that depicts how well potential throughput $\mathcal{P}$ correlates with the assigned priority vectors $\mathbf{p}$. The mixes depicted are directly ported from Section V-A. We also sorted workloads from least to most computationally demanding ones. We employ the Pearson correlation coefficient $\mathbf{r} \in [-1, 1]$ to quantify the relationship between $\mathcal{P}$ and $\mathbf{p}$.

In Figure 7, we observe that for mixes of 3 concurrently executing DNNs, RankMap$_\mathcal{D}$ yields mappings that highly satisfy the provided priority vectors $\mathbf{p}$, resulting in 0.85 average $\mathbf{r}$. Naturally, RankMap$_\mathcal{D}$ has more difficulty satisfying the priorities assigned as the computational demands of the workloads increase. This is also evident with mixes of 4 concurrently executing DNNs, where Mix-5 and Mix-6 score below 0.5 $\mathbf{r}$. This is because all devices begin to saturate, and for RankMap$_\mathcal{D}$ to prevent starvation for all DNNs, it deviates from the given priority vector. Finally, the platform saturates completely in several mixes of 5 DNNs. Even under these extreme conditions, RankMap$_\mathcal{D}$ maintains positive $\mathbf{r}$ values. Hence, our framework strives for a balanced mapping that considers throughput, prioritization constraints, and starvation prevention, even in highly resource-constrained settings.

RankMap$_\mathcal{S}$ operates based on user-defined priorities. To demonstrate its efficacy in balancing throughput optimization while satisfying priority constraints and preventing throttling effects across all DNNs in a workload, we present a scenario illustrated in Figure 8. This scenario comprises a workload of four concurrent DNNs: **(i)** MobileNet-V2; **(ii)** ShuffleNet; **(iii)** AlexNet; and **(iv)** SqueezeNet. The experiment shows dynamic changes in user priorities across multiple stages. In the initial stage, the user-defined priority vector $\mathbf{p}1$ is set to $(0.7, 0.1, 0.1, 0.1)$, with RankMap$_\mathcal{S}$ constructing a mapping that adheres to these priorities. Then, in stage two, the user changes the priorities to $\mathbf{p}2 = (0.1, 0.7, 0.1, 0.1)$, shifting the highest priority to ShuffleNet, a less computationally demanding network compared to MobileNet-V2. RankMap$_\mathcal{S}$ adapts to this change, satisfying the new priority configuration while ensuring no DNN experiences starvation. This adaptive behavior persists through the remaining stages of priority shifts to the other two DNNs. The dashed grey lines indicate the time it takes for RankMap$_\mathcal{S}$ to identify the appropriate mappings, detailed further in Section V-D.

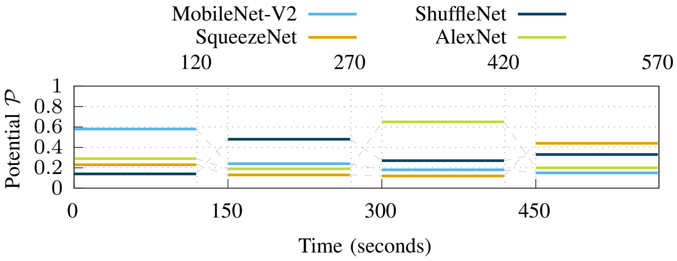


Fig. 8: Potential Throughput $\mathcal{P}$ for different priority vectors $\mathbf{p}$ using RankMap$_\mathcal{S}$ for the mix of: **(i)** MobileNet-V2; **(ii)** SqueezeNet-V1; **(iii)** ShuffleNet; and **(iv)** AlexNet.

D. *Run-time performance evaluation*

The Baseline manager is the fastest in response time as it directly maps all DNNs to the GPU, but fails to utilize the platform's heterogeneity, leading to suboptimal throughput, prioritization, and increased starvation risk. MOSAIC, slightly faster than ODMDEF (both around 1 second), quickly causes device saturation as demonstrated in sections V-A and V-B, resulting in low throughput and starvation, which shows the limitations of simplistic designs. In contrast, the GA is the slowest, requiring evaluations for each chromosome on the Orange Pi 5 for every generation, and it cannot use past data to predict or adapt to new workloads. OmniBoost and our RankMap both show a run-time of about 30 seconds, suggesting that RankMap achieves an optimal balance of run-time performance and efficient workload management, supporting high throughput, prioritization, and starvation prevention.

VI. Conclusion

Modern application workloads, involving multiple concurrent DNNs, pose significant challenges for heterogeneous embedded systems focused on optimizing throughput and managing priorities without causing starvation. This paper presents RankMap, a scalable, efficient manager that enhances system performance. RankMap boosts average throughput by up to ×3.6 and improves resource allocation effectiveness by up to ×57.5 compared to existing solutions. Importantly, it prevents application starvation across multi-DNN workloads.

Acknowledgments

This work is supported by grant NSF CCF 2324854.

References

[1] C.-J. Wu *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[2] C. Hsieh *et al.*, "Surf: Self-aware unified runtime framework for parallel programs on heterogeneous mobile architectures," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019.

[3] D. Kang *et al.*, "Scheduling of deep learning applications onto heterogeneous processors in an embedded device," *IEEE Access*, 2020.

[4] A. Karatzas and I. Anagnostopoulos, "Omniboost: Boosting throughput of heterogeneous embedded devices under multi-dnn workload," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[5] C. Wang, Y. Bai, and D. Sun, "Cd-msa: Cooperative and deadline-aware scheduling for efficient multi-tenancy on dnn accelerators," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[6] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[7] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 82–95.

[8] J. Lin, H. Yin, W. Ping, Y. Lu, P. Molchanov, A. Tao, H. Mao, J. Kautz, M. Shoeybi, and S. Han, "Vila: On pre-training for visual language models," 2023.

[9] A. S. Developers. (2023) Arm NN: The easy way to deploy edge ML. Arm Community. [Online]. Available: https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/arm-nn-the-easy-way-to-deploy-edge-ml

[10] S. Wang *et al.*, "High-throughput cnn inference on embedded arm big. little multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[11] E. Aghapour, D. Sapra, A. Pimentel, and A. Pathania, "Arm-co-up: Arm co operative u tilization of p rocessors," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[12] J. S. Jeong *et al.*, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," in *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, 2022.

[13] H. Kwon *et al.*, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[14] A. Das, E. Russo, and M. Palesi, "Multi-objective hardware-mapping co-optimisation for multi-dnn workloads on chiplet-based accelerators," *IEEE Transactions on Computers*, 2024.

[15] O. Spantidi *et al.*, "Targeting dnn inference via efficient utilization of heterogeneous precision dnn accelerators," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[16] A. Archer, M. Fahrbach, K. Liu, and P. Prabhu, "Pipeline parallelism for dnn inference with practical performance guarantees," *arXiv preprint arXiv:2311.03703*, 2023.

[17] M. Han *et al.*, "Mosaic: Heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2019.

[18] C. Lim and M. Kim, "Odmdef: on-device multi-dnn execution framework utilizing adaptive layer-allocation on general purpose cores and accelerators," *IEEE Access*, vol. 9, pp. 85 403–85 417, 2021.

[19] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, 2017.

[20] T. Mikolov *et al.*, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[21] D. Yang, S. Liu *et al.*, "Hifi-codec: Group-residual vector quantization for high fidelity audio codec," *arXiv preprint arXiv:2305.02765*, 2023.

[22] A. Vaswani, N. Shazeer *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[23] Z. Shen, M. Zhang *et al.*, "Efficient attention: Attention with linear complexities," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2021, pp. 3531–3539.

[24] M. Świechowski *et al.*, "Monte carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, 2022.

[25] ARM. (2017) Arm compute library. [Online]. [Online]. Available: https://www.arm.com/technologies/compute-library