Balancing Throughput and Fair Execution of Multi-DNN Workloads on Heterogeneous Embedded Devices

Andreas Karatzas, Iraklis Anagnostopoulos, *Member, IEEE*

Abstract—The rise of Deep Neural Networks (DNNs) has resulted in complex workloads employing multiple DNNs concurrently. This trend introduces unique challenges related to workload distribution, particularly in heterogeneous embedded systems. Current run-time managers struggle to efficiently utilize all computing components on these platforms, resulting in two major problems. First, the system throughput deteriorates due to contention on the computing resources. Second, not all DNNs are affected equally, leading to inconsistent performance levels across different models. To address these challenges, we introduce FairBoost, a framework for efficient and fair multi-DNN inference on heterogeneous embedded systems. FairBoost employs Reinforcement Learning (RL) to efficiently manage multi-DNN workloads. Additionally, it incorporates a novel numerical representation of DNN layers via a Vector Quantized Variational Auto-Encoder (VQ-VAE). Finally, it enables knowledge transfer to similar heterogeneous embedded systems without retraining and/or fine-tuning. Experimental evaluation of FairBoost over 18 DNNs and various multi-DNN scenarios shows an average throughput/fairness improvement of $\times 3.24$. Additionally, FairBoost facilitates knowledge transfer from the initial platform, Orange Pi 5, to a new system, Odroid N2+, without any retraining or fine-tuning achieving similar gains.

Index Terms—Heterogeneous embedded systems, multi-DNN workload, reinforcement learning, throughput optimization.

1 Introduction

The advent of Deep Neural Networks (DNNs) has revolutionized the field of embedded devices, leading to the development of various embedded applications such as digital assistants and AR/VR services [1]. However, these advancements come with the challenge of deploying these highly resource-demanding applications on embedded devices, often characterized by limited computational capabilities. To address this computational barrier, platform heterogeneity has emerged as a promising solution to augment system performance. Nonetheless, this architectural heterogeneity is often underutilized, introducing new barriers in edge inference [2], [3]. Existing deep learning frameworks utilize the CPU or the GPU but not both; thus, they cannot capitalize on the underlying heterogeneity. This limitation arises from programmability issues between different computing elements. Therefore, to efficiently utilize the inherent heterogeneity, deep learning frameworks must be designed to leverage all available computing components optimally.

Furthermore, many modern services employ multiple DNN applications concurrently to deliver more sophisticated and complex services [1], [4]. For instance, in the AR/VR domain, applications require a variety of diverse tasks, such as object detection, object classification, hand tracking, hand pose, and depth estimation [5], [6]. Accordingly, heterogeneous embedded devices are required to manage multi-DNN workloads, each presenting a unique computational profile. This scenario introduces additional workload management challenges as resources must be collaboratively managed to uphold the required quality of service [7]. Naturally, assigning multiple DNNs solely to high-performance processing elements, such as GPUs, can result in up to $\times 4.6$ performance reduction [8].

Another aspect for evaluating system performance in multi-program workload is *fairness* in execution. Fairness provides a comprehensive understanding of system behavior, which throughput alone might not fully capture [9]. In multi-DNN workloads, considering fairness (alongside throughput) has become particularly important due to the interference and contention over shared resources [10]–[12]. As the performance degradation of concurrently executing DNNs is not balanced (e.g., some DNNs suffer more significant throughput drop than others), the system becomes unpredictable, significantly affecting QoS [13]–[15]. The concept behind fairness, in this case, is to ensure that all DNNs have a sufficient progress rate of execution. The problem of balancing throughput and fairness for multi-DNN workloads becomes increasingly prominent as state-of-art DNNs grow in complexity, thereby augmenting the uneven throughput among different DNNs. *A way to increase the overall throughput and fairness of multi-DNN workloads is to utilize all the computing components synergistically*. Conversely, collaboratively utilizing computationally diverse components (e.g., big.LITTLE CPUs, embedded GPUs) can notably improve the overall system throughput [16].

A promising method to capitalize on the system's architectural heterogeneity is DNN partitioning, a process where different parts of a DNN are allocated to different computing components. However, due to the expansive search space size, finding the optimal split points of DNNs to improve system throughput and fairness is nontrivial.

- *A. Karatzas and I. Anagnostopoulos are with the School of Electrical, Computer and Biomedical Engineering, Southern Illinois University, Carbondale, IL 62901 USA.*

Corresponding author: Andreas Karatzas (andreas.karatzas@siu.edu).

To distribute the multi-DNN workload efficiently across the system to increase throughput and fairness, run-time managers must: (**i**) group adjacent DNN layers into stages [17] to create efficient pipelines at run-time, and (**ii**) efficiently map the compiled sub-networks across the available computing components. Nonetheless, modern DNNs encompass multiple layers, and newer heterogeneous embedded devices comprise more diverse computing components. Therefore, identifying the optimal partition points for a set of multi-DNN workloads while simultaneously increasing throughput and fairness becomes significantly complex [2]. The complexity increases even more when trying to reduce the decision-making time for partitions during run-time.

In this work, we propose FairBoost, a framework for efficient and fair multi-DNN inference on heterogeneous embedded systems. FairBoost leverages DNN layer partitioning to boost the overall system throughput and fairness of multi-DNN workloads deployed on heterogeneous embedded devices. It introduces multidimensional layer representation to accurately predict throughput and facilitate more effective knowledge transfer via a Vector Quantized Variational Auto-Encoder (VQ-VAE). FairBoost significantly enriches the layer representation with a highly scalable codebook composed of distributed embedding vectors, each signifying the computational profile of individual DNN layers. Finding how to synergistically partition multiple DNNs targeting both objectives is a task that spawns completely new challenges. To that end, FairBoost features knowledge transfer, allowing it to be ported from one board to another without retraining or fine-tuning.

Our primary contributions are outlined as follows: ❶ **Layer partitioning to boost the overall system throughput and fairness**: FairBoost leverages DNN layer partitioning to boost both the overall system throughput and fairness of multi-DNN workloads deployed on heterogeneous embedded devices; ❷ **Multi-dimensional layer representation**: We introduce multi-dimensional layer representation to accurately predict throughput and facilitate more effective knowledge transfer; ❸ **Latent-space encoding**: We implemented a Vector Quantized Variational Auto-Encoder (VQ-VAE), which reduces the dimensionality of the data and, consequently, the computational load; and ❹ **Formulated the environment for the RL agent**: We developed the environment for the RL agent to facilitate effective training, including constructing the states and actions within the environment and devising a reward function that is designed to *co-optimize* overall system throughput and fairness simultaneously.

2 Motivation

To demonstrate the importance of throughput and fairness co-optimization in concurrent *multi-DNN execution*, we will consider the demanding AR/VR application composed of 4 sub-DNNs presented in [4]. The application consists of: (**i**) YOLO [18] for identifying objects; (**ii**) FaceNet [19] for detecting faces; (**iii**) AgeNet [20] for discerning the age; and (**iv**) GenderNet [21] for predicting the gender. If we optimize only for system throughput, our manager will likely prioritize less demanding DNNs, such as FaceNet, thus boosting the workload throughput. However, this will also negatively impact the performance of more resource-intensive DNNs, like YOLO. This problem of multi-DNN services was also emphasized at CES 2024 during the presentation of Qualcomm's next-gen XR chip, which supports up to a 4.3K resolution per eye at 90 frames per second [22].

For our own motivational example, we utilized the Orange Pi 5 board [23], which features a Mali-G610 GPU and big.LITTLE CPUs with a quad-core Cortex-A76 running at 2.4GHz and a quad-core Cortex-A55 at 1.8GHz. Regarding the multi-DNN workload, we selected four diverse and widely utilized DNNs: (**i**) AlexNet [24], (**ii**) MobileNet V2 [25], (**iii**) ResNet-50 V2 [26], and (**iv**) ShuffleNet [27].

To quantify the efficiency of DNN partitioning, we define: ❶ *throughput* $\mathcal{T}$ of a multi-DNN workload as the average number of DNN inferences per second; and ❷ *fairness*, using the Jain's fairness index [28]. Mathematically:

$$\mathcal{T} = \frac{1}{N}\sum_{i=1}^{N} t^{i}_{current}, \qquad \mathcal{J} = \frac{\left(\sum_{i=1}^{N} \frac{t^{i}_{current}}{t^{i}_{ideal}}\right)^{2}}{N \cdot \sum_{i=1}^{N} \left(\frac{t^{i}_{current}}{t^{i}_{ideal}}\right)^{2}} \qquad (1)$$

where N depicts the total number of DNNs in the workload and $t^{i}_{current}$ is inferences per second of DNN i, and t^{i}_{ideal} represents the throughput of DNN i when executed alone on the embedded GPU. The closest $\mathcal{J}$ is to 1, the more fair the system is. Fairness correlates to how multiple DNNs affect each other when running concurrently. The throughput of a single DNN decreases when other DNNs run on the GPU simultaneously, compared to when it runs alone.

First, we mapped all DNN models on the GPU, as it is traditionally favored for its superior computing capabilities. This gave us a baseline performance metric to compare with our upcoming experimental configurations. Then, we created 200 unique DNN mappings from the same multi-DNN workload. We randomly split the layers of the DNNs among the three available computing components (big CPU cluster, LITTLE CPU cluster, and GPU) and arbitrarily chose one for each pipeline stage. An example of a mapping is: (**i**) AlexNet: first 2 layers on big CPU cluster, next 2 layers on LITTLE CPU cluster, and the remaining ones on the GPU; (**ii**) MobileNet V2: first 5 layers on GPU, next 3 layers on the LITTLE CPU cluster, and the remaining ones on the big CPU cluster; (**iii**) ResNet-50 V2: first 10 layers on big CPU cluster and the remaining on the LITTLE CPU cluster; (**iv**) ShuffleNet: all layers on the GPU. For these four selected DNNs, the total number of possible mapping configurations is more than $4{*}10^{9}$. So, we only show 200 random mappings for demonstration and reading comprehension reasons.

Figure 1 depicts the outcome for the 200 different setups in terms of throughput (green points) and fairness (orange points). For simplicity and readability, the displayed setups are sorted with respect to the throughput $\mathcal{T}$ (green dots) normalized over the baseline (all DNNs are executed on the GPU). We also show the achieved corresponding fairness index $\mathcal{J}$ (orange dots) for each setup. We observe that while the baseline achieves better throughput than most setups, there are approximately 20% different setups (~ 40 points illustrated as green dots above the blue line) that achieve up to 90% better throughput in the best case. Moreover, 57% of the configurations demonstrated better fairness when compared to the baseline. The task of identifying an optimal

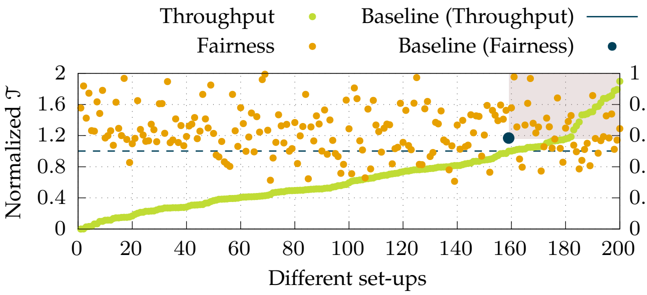


Fig. 1: Normalized throughput and fairness for 200 random setups on the Orange Pi 5 board. For each setup, we randomly split the layers between CPU and GPU. Baseline is the case in which all DNNs are executed on the GPU.

solution, however, presents a distinct challenge. The ideal configuration is one that strikes an effective balance between throughput and fairness. *The orange points in the shaded rectangle area correspond to configurations that surpassed the baseline in both throughput and fairness.*

The vastness of the solution space and the cost of exploring it are noteworthy considerations. For this particular example only, the total number of combinations is $3^8 + 3^{20} + 3^{18} + 3^{18} \approx 4 * 10^9$, where $\{8, 20, 18, 18\}$ are the valid partition points per DNN in our workload and 3 is the number of computing components (i.e., big CPU cluster, LITTLE CPU cluster, and GPU). Nevertheless, this number corresponds only to these particular DNNs, and if we expand our choices and consider more DNN architectures, those combinations yield a design space in the order of tens of billions. Hence, the task of executing multiple DNNs on heterogeneous embedded systems transposes into a well-known NP-hard problem [2]. This renders simplistic strategies such as a greedy search inadequate and necessitates deploying more sophisticated exploration techniques.

3 Related Work

Towards the efficient utilization of heterogeneity in embedded systems, the works in [29], [30] utilize the inter-layer parallelism of DNNs, but evaluate only for throughput rather than fairness and hence often yield unfair pipelines. In [13], [31], fairness is defined as the normalized progress, representing the slowdown of concurrent multi-DNN execution compared to each DNN's isolated execution. Additionally, the works in [10], [12] use a fairness metric to measure the equal progress under multi-DNN workloads compared to the single DNN's isolated execution. Fairness ranges between 0, indicating no fairness, i.e., at least one of the concurrently executed DNNs starves, and 1, indicating perfect fairness, i.e., all concurrently executed DNNs make equal progress. However, none of these methods consider DNN partitioning and multi-objective optimization. The approach in [16] builds upon the observation that the execution time of a CNN layer is linearly correlated to the dimension of matrices involved in the layer operations. The authors in [32] present another framework that utilizes the GPU for the model's most computationally intensive layers.

Recently, research around DNN partitioning has been utilized to optimize the system quality of service [33]. The framework in [3] exploits the range of mobile heterogeneous compute units to optimize the system's throughput. The latency estimation model in [34] attempts to find DNN pipelines that optimize the utilization of heterogeneous compute units. Authors in [1] propose a design space exploration algorithm for DNN pipeline configuration that defaults to an exhaustive search over all candidate solutions. BAND [35] introduces a manager that attempts to find the DNN sub-graphs with common computing component operators in order to group them. Authors in [7] utilize a look-up table (LUT) to help managers make run-time decisions. MASA [4] is a multi-DNN manager with a layer-by-layer processing sequence. The authors in [36] introduce DART, a framework that uses a database to encapsulate all potential DNN pipeline combinations to generate a mapping. Moreover, MOSAIC [37] employs DNN partitioning to distribute DNN workloads. Its core is a linear regression model trained in single DNN scenarios, building upon the correlation between each layer's input feature map dimensions and its corresponding compute requirements. Nonetheless, this approach does not consider all DNNs in the workload while configuring the partition points. Thereby, MOSAIC maps most of the workload to the highest-performing computing element, leading to low throughput. ODMDEF [38] is another framework that uses linear regression and k-nearest neighbors to yield pipelines for multi-DNN workloads. However, this approach requires a large number of samples to reach an acceptable model accuracy. Furthermore, it does not consider fairness in managing the models, leading to unfair multi-DNN pipelines. Authors in [2] propose a framework that uses an evolutionary algorithm to tackle the large search space. Nonetheless, it requires retraining as the populations from previous multi-DNN workloads become irrelevant for succeeding workloads. Moreover, the fitness function employed in the fitting procedure is not designed to ensure fairness within the DNNs.

TABLE 1: Qualitative comparison between existing state-of-the-art works and FairBoost. The major differentiators of our work compared to the state-of-art are marked with †.

	Baseline	RR	MOSAIC [37]	ODMDEF [38]	GA [2]	**FairBoost**
Features						
Single-DNN Workloads	✓	✓	✓	✓	✓	✔
Multi-DNN Workloads	-	✓	-	-	✓	✔
Layer Partitioning	-	-	✓	✓	✓	✔
Multi-dimensional Layer Representation†	-	-	-	-	-	✔
Multi-objective Co-Opt.†	-	-	-	-	-	✔
Latent Space Encoding†	-	-	-	-	-	✔
Fast Training	-	-	✓	✓	-	✔
Transferable	-	-	-	-	-	✔
Metrics						
Throughput	-	-	✓	✓	✓	✔
Fairness	-	✓	-	-	-	✔

Our key differences from the state-of-the-art are manyfold: (**i**) we facilitate efficient design space exploration by employing a Reinforcement Learning (RL) agent to capture

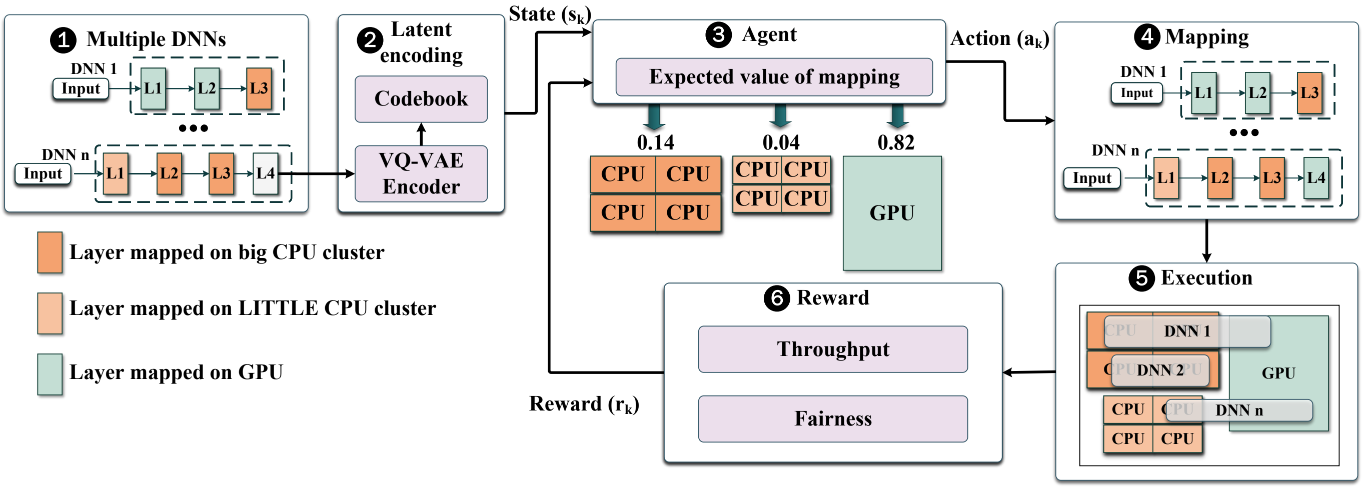


Fig. 2: Architectural overview of FairBoost.

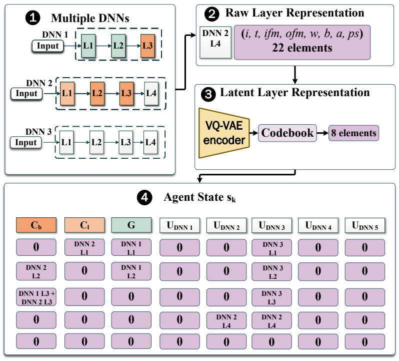
and extrapolate the intricate interactions between the DNNs and the underlying computing components; (**ii**) we develop a highly scalable codebook of distributed embedding vectors, with each vector representing the computational profile of an individual DNN layer; (**iii**) FairBoost applies a tailored reward function to co-optimize throughput and fairness; (**iv**) FairBoost supports knowledge transfer from one platform to another without any retraining and/or fine-tuning; and (**v**) we effectively reduce training complexity by leveraging action masking, eliminating invalid actions from the action space, and enabling the RL agent to converge faster. A qualitative comparison is presented in Table 1.

4 Proposed Framework

FairBoost is designed for heterogeneous embedded systems. For our analysis, we focus on a conventional embedded system architecture that includes the following computing components: 1) a high-performance CPU cluster (big CPU); 2) a low-performance CPU cluster (LITTLE CPU); and 3) an embedded GPU. However, it's worth noting that FairBoost is scalable and can support a greater variety of computing resources. Figure 2 provides an overview of FairBoost's architecture, which consists of six steps. These steps are iteratively executed for each DNN layer of a multi-DNN workload. **Step ❶:** In the first step, we decompose the given models layer-wise. After identifying the layers of each DNN, we use their characteristics to encompass elements such as the layer type (e.g., convolution layer, fully connected layer, etc.), the dimensions of the input and output feature maps, and the dimensions of the weights tensor, and create rich numerical DNN layer representations. **Step ❷:** Once the input workload is numerically formulated, we employ the encoder module of a VQ-VAE model [39] to transpose the raw data into rich latent vectors. This encoder compresses and distributes the information uniformly in latent vectors. Furthermore, this process significantly reduces the dimensionality of the input space, subsequently minimizing our framework's overall complexity. **Step ❸:** We utilize the latent representation of the input workload as input to our workload manager. This manager is a reinforcement learning agent powered by a ResNet-9 Convolutional Neural Network (CNN). This module estimates the layer's performance on the different computing components (Q-values). **Step ❹:** We use the estimated Q-values to map the layer that is being processed in step k. The action α_k with the highest Q-value represents the computing component used for that layer's execution. **Step ❺:** In this step, we execute the workload configured with respect to the agent's action trajectory on the actual embedded device. As output, we get the corresponding throughput. **Step ❻:** We use throughput and fairness to provide feedback to our agent for any given scenario. This feedback r_k allows us to assess the agent's performance and train it towards creating mappings that perform better in terms of both throughput and fairness under multi-DNN workloads. Once the training is completed, FairBoost supports event-driven execution. When one or multiple inputs trigger the execution of multiple DNNs, FairBoost analyzes the computational characteristics of the DNN layers and decides the splitting and mapping of them on the available computing resources of the platform in order to boost throughput and, at the same time, improve fair execution. These computational characteristics are extracted by loading the models' graphs generated during model compilation (before the actual execution). At this point, it is important to mention that FairBoost can handle unseen DNNs due to the VQ-VAE's robustness, which encodes any layer's important information and then returns the closest embedding from its codebook.

4.1 DNN decomposition

In this section, we formulate the input for our Vector Quantized Variational Auto-Encoder (VQ-VAE). We introduce a resilient, flexible numerical representation that depicts any given DNN. FairBoost leverages DNN inter-layer parallelism to partition any DNN model, necessitating a layer-level input representation. Taking advantage of the linear correlation between the dimensions of a layer's weights tensor and its computational complexity [37], we compile a vector that encapsulates specifications regarding multiple



C_b	C_l	G	$U_{DNN\,1}$	$U_{DNN\,2}$	$U_{DNN\,3}$	$U_{DNN\,4}$	$U_{DNN\,5}$
0	DNN 2 L1	DNN 1 L1	0	0	DNN 3 L1	0	0
DNN 2 L2	0	DNN 1 L2	0	0	DNN 3 L2	0	0
DNN 1 L3 + DNN 2 L3	0	0	0	0	DNN 3 L3	0	0
0	0	0	0	DNN 2 L4	DNN 2 L4	0	0
0	0	0	0	0	0	0	0

Fig. 3: Input formulation for our Rainbow-DQN.

tensors describing any given DNN layer. Contrary to previous methods [2], [8], our approach formulates the input into multiple dimensions, thereby enriching the layer representation and enabling more efficient multi-DNN pipelines in the next steps. For each DNN j, our proposed input 22-dimensional vector comprises each layer's characteristics. Furthermore, **ifm**, **ofm**, and **w** are 4-dimensional tensors, that correspond to: (**i**) The batch size; (**ii**) The number of channels; (**iii**) The feature map height; and (**iv**) The feature map width. Finally, **ps** is a 6-dimensional vector representing the layer's pad-stride information.

$$\mathbf{L}_i^j = \left(\underset{\text{Layer index of DNN } j}{i} \quad \overset{\text{Layer type}}{\mathbf{t}} \quad \underset{\text{Input feature map}}{\mathbf{ifm}} \quad \overset{\text{Output feature map}}{\mathbf{ofm}} \quad \underset{\text{Weights tensor}}{\mathbf{w}} \quad \overset{\text{Number of biases}}{\mathbf{b}} \quad \underset{\text{Type of activation}}{\mathbf{a}} \quad \overset{\text{Pad-stride information}}{\mathbf{ps}} \right) \tag{2}$$

The rationale behind this approach is to condition our agent on both the complexity and size of the DNNs in the workload. The input and feature map (ifm), output feature map (ofm), weights tensor (w), number of biases (b), and pad-stride information (ps) allow the agent to know the complexity of a layer. Concurrently, attention to the layer index (i) informs the agent of the depth of the DNN.

4.2 Latent layer representation

FairBoost utilizes a Vector Quantized Variational Auto-Encoder (VQ-VAE) [39], compiled by 1D convolutional layers. In this study, our analysis focuses on the following subset of parameters: (**i**) the encoder module, and; (**ii**) the codebook associated with the learned latent space. Figure 3 depicts the input formulation for our Rainbow-DQN.

The encoder transposes the raw numerical representations L_i^j of layer i in DNN j into a latent representation Z_i^j. More specifically, FairBoost applies quantization to the distribution of $q(z|x)$, employing discrete latent variables to formulate the specified multi-DNN workload. This strategy not only effectuates significant dimensionality reduction but also amplifies the representation of valuable vector features in the latent space [40]. This refined approach significantly improves the precision and efficiency of the workload manager, thereby underlining the integral role of the VQ-VAE module in our framework. The mapping process from the decomposed vector layer representation L_i^j into a sequence of discrete latent variables Z_i^j is facilitated by: (**i**) a 1D convolutional layer that extracts the high-level features, and; (**ii**) a block of 9 1D convolutional residual layers that capture the lower-level features. Within the multi-DNN workload, the raw numerical representation L_i^j of layer i of DNN j is iteratively fed into our encoder. The output of this process is a latent matrix with dimensions $\left(\sum_j^M |L^j|\right) \times 8$, where M denotes the total number of models within the multi-DNN workload, $|L^j|$ represents the number of layers in the j-th DNN of the workload, and 8 signifies the dimensions of the latent space. The choice of 8 for the latent space dimensions was not arbitrary. We conducted extensive experimentation with various latent vector dimensions, as demonstrated later in Section 5. Our observations suggest that a dimension of 4 results in an excessively compressed representation, signifying a substantial loss of information during the compression process, which is later translated into a performance drop on the agent's part. Conversely, a dimension of 16 introduces additional computational requirements for handling extra parameters, but the corresponding optimization benefits are minimal. Therefore, 8 was determined to be the most appropriate choice, striking a balance between sufficient data encoding accuracy and computational efficiency.

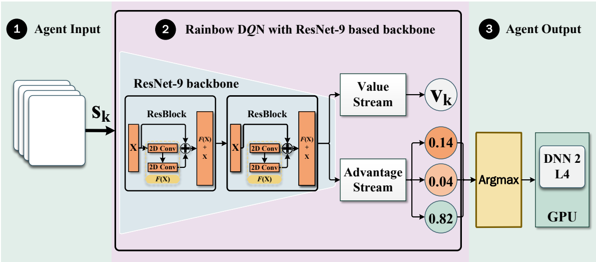


Fig. 4: High-level overview of our Rainbow-DQN.

The VQ-VAE plays a crucial role in the dimensionality reduction of our framework, providing a vital benefit in terms of computational efficiency. Notably, the VQ-VAE greatly prunes the agent's number of Multiply-Accumulate (MAC) operations. Without the VQ-VAE, the agent requires $\approx 14B$ MAC operations for inference. However, implementing VQ-VAE brings this number down to $\approx 2B$. This equates to more than $\times 7.5$ reduction in computational complexity, freeing up significant processing power, thus enabling a more efficient and compute-friendly framework.

4.3 Agent

In this work, we employ a Rainbow-DQN [41] agent to find optimal mappings. We assume that there is a discrete number of computing components for any given embedded device. In our case, these elements are the big.LITTLE CPU clusters and the embedded GPU. Hence, our problem operates within a discrete action space, which renders a value-based agent suitable. Consequently, we utilize the Rainbow algorithm, a state-of-the-art Q-learning algorithm. Figure 4 depicts a high-level overview of our Rainbow-DQN agent.

We develop a lightweight 9-layer convolutional neural network (CNN) model that inherits from the widely

Algorithm 1 Fairness and throughput co-optimization

```
1: function $\textsc{Reward}(\overrightarrow{T_{current}}, \overrightarrow{T_{ideal}}, \alpha, \beta, \gamma)$
2:     $ratios \leftarrow \textsc{StdDev}(^{i=0}_{N}(\frac{t^i_{current}}{t^i_{ideal}}) * 100)/50$
3:     $fairness \leftarrow 2 * \textsc{Sigmoid}(-\alpha * (ratios - 0.5)) - 1$
4:     $t_{achieved} \leftarrow \frac{\textsc{sum}(^N_{i=0} t^i_{current})}{\textsc{sum}(^N_{i=0} t^i_{ideal})}$
5:     $throughput \leftarrow 2 * \textsc{Sigmoid}(\gamma * (t_{achieved} - 0.5)) - 1$
6:     $r \leftarrow (1 - \beta) * fairness + \beta * throughput$
7:     return $r$
8: end function
```

adopted ResNet DNN family [42]. This model, while maintaining the core principles of the ResNet architecture, is designed with an emphasis on computational efficiency and training speed-up. To that end, our custom ResNet-9 is characterized by a minimal size of $\approx 11M$ trainable parameters, thereby ensuring our framework's ability to learn the required data patterns rapidly. Finally, in our model's architecture, we incorporated 512 neurons within the noisy linear layers to ensure the propagation of valuable low-level features detected by the convolutional layers.

4.4 Environment

To deploy the Rainbow-DQN agent described in subsection 4.3, we define an environment composed of **states**, **actions**, and **rewards**. The agent operates in this environment by selecting an **action**, which is equivalent to choosing a computing component to process a specific DNN layer at each time step k. The agent receives an observation matrix $\mathcal{S}$, structured such that columns are configured into eight-element block matrices:

$$\mathcal{S}_k = \begin{pmatrix} C_b & C_l & G & U_{DNN_1} & \dots & U_{DNN_5} \end{pmatrix} \tag{3}$$

where C_b, C_l, and G represent the big CPU cluster, the LITTLE CPU cluster, and the embedded GPU, respectively. Each U_{DNN_i} represents a queued DNN, where i varies from 1 to 5. The matrix can be further extended to support more DNNs, but due to the limited computational resources of our embedded device, we restrict our analysis to five DNNs. Figure 3 depicts this observation matrix in Step ❹. Matrix $\mathcal{S}_k$ provides a snapshot of the environment **state** at any specific time k. Each row corresponds to the latent representation Z^j_i of layer i of DNN j. Each column corresponds to either a computing component or a queued DNN. This encoding gives our agent a holistic view of the environment's current state, enabling more efficient decision-making.

To train our agent in the aforementioned environment, we conceptualize a reward function that continuously integrates both aspects and synergistically optimizes them. This reward function optimizes a pair of objectives: (**i**) maximize average throughput in a multi-DNN workload; and (**ii**) maximize the fairness in resource distribution amongst the DNNs. The proposed reward function is described in Algorithm 1. Optimizing for a composite objective requires careful mathematical formulation because of the differences in measurement and scale amongst the utilized metrics, i.e., throughput and fairness. To that end, we incorporated the ideal throughput for each DNN, which can be calculated by running any given model for a small period of time alone on the GPU, the highest-performing computing component on our platform. By introducing the ideal throughput in tandem with the currently achieved throughput of any given mapping, we were able to calculate ratios of performance for each DNN and thus guarantee a vector of numbers in the range of $[0, 1]$. After normalizing our values, we are ready to quantify throughput and fairness: ❶ **For throughput**: We sum the achieved throughput for each DNN in the workload given the mapping and divide it by the sum of ideal throughputs of these DNNs. We pass the result of line 4 into the expression in line 5. In this expression, we first zero-mean t_{achieved}, then scale it using γ, and finally pass it through `sigmoid`. ❷ **For fairness**: We can evaluate the agent's performance in terms of fairness using the standard deviation of $\mathcal{Q}$. The reason behind opting for `std` instead of Jain index is that we observed that Jain index is more aggressive towards fairness than `std`, which poses a significant challenge for the agent to eventually converge. By utilizing a metric more "robust" to unfairness, our agent could quickly detect the hidden patterns in its experiences regarding fairness. ❸ **Co-optimization of throughput and fairness**: If we opt for a metric that evaluates the agent's performance with respect to throughput, then our agent will yield mappings that completely disregard fairness. Similarly, if we only leverage fairness to evaluate our agent's performance, then the resulting mappings would completely disregard throughput, leading to a suboptimal multi-DNN manager. For example, a 4-DNN mapping that yields $[0, 0, 0, 0]$ throughput is amongst the fairest mappings that could exist for that workload, but starves all the DNNs in the workload. To that end, we synergistically combine throughput and fairness into a weighted sum in line 6 of our algorithm. An energy distribution of our composite reward function is given in Figure 5.

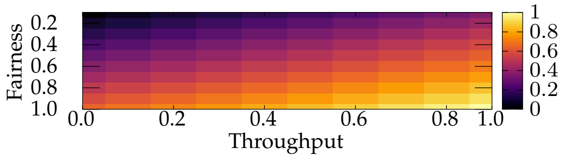


Fig. 5: Heatmap of our composite reward function.

4.5 Training acceleration

Our agent rapidly converges within $20K$ training steps, representing $\times 1,000$ acceleration in training compared to a conventional reinforcement learning (RL) agent in an Atari 2600 environment, which necessitates over $20M$ steps to converge [41]. We achieved this remarkable enhancement by applying **action masking** [43], which efficiently refines the agent's focus onto relevant data patterns. Action masking is the process of zeroing the logits associated with impossible actions, ensuring that such actions are not selected during the decision-making process. In the case of a value-based algorithm, an action is selected based on the maximum estimated value of the action-value function $Q(s, a)$. Mathematically, this is formulated in Equation 4.

$$a = \arg\max_{a \in A} Q(s, \cdot) \tag{4}$$

where, s represents the current state, A the action space, and a the action taken. The agent selects the action that maximizes the Q value for the given state. When the action mask is applied, the Q-values associated with the impossible actions are set to negative infinity ($-\infty$). This essentially removes these actions since their Q-values can never be the highest. Consequently, this leads to more efficient training as the agent focuses on viable actions [44].

Action masking results in efficient data transfer minimization in multi-DNN workloads. As aforementioned, our heterogeneous embedded device environment comprises three distinct computing components: (**i**) a big CPU cluster; (**ii**) LITTLE CPU cluster; and (**iii**) an embedded GPU. The available actions represent the selection of these devices. The action masking strategy we employ is guided by the sequence of device selections made by the agent.

The mathematical formulation for building the mask in our environment is as follows. Let $D = \{d_1, d_2, d_3\}$ denote the set of devices, with d_1 representing the big CPU cluster, d_2 the LITTLE CPU cluster, and d_3 the embedded GPU. Let a_k denote the action at time k, which is equivalent to the device selected by the agent at that time. If the agent decides to utilize the same device at time $k+1$ as in time k, i.e., $a_{k+1} = a_k$, then the action mask, M, remains the same as in the previous step. Mathematically, this is represented as $M_{k+1} = M_k$. On the other hand, if the agent opts to switch to a different device at time $k+1$, i.e., $a_{k+1} \neq a_k$, then the action mask is updated to exclude the device selected at time k. Mathematically, this can be represented as $M_{k+1} = D \backslash a_k$.

Another important mechanism for training acceleration is our efficient memory mechanism. This mechanism is designed to retain records of all previously encountered queries. By preserving each DNN mapping, the system can leverage identical or permutation-equivalent mappings in future steps. In such instances, the memory efficiently retrieves the already inferred results. The introduction of this memory mechanism contributes significantly to the training speed-up. With this optimization, there is no longer a constant requirement to submit the agent's compiled DNN workload to the utilized embedded device for evaluation. Instead, we can utilize pre-computed system throughput results and add small amounts of white noise to circumvent the potential overestimation of Q-values. This practice demonstrates the principles of dynamic programming by intelligently reusing previously computed results to avoid redundant computations. Consequently, this mechanism reduces the training time by $\times 5$.

4.6 Knowledge transfer

The time-intensive training process involved in our agent's design necessitates a mechanism for the seamless transfer of knowledge from one platform to another. FairBoost utilizes OpenCL and the ARM Compute Library [45]. This inherently aligns our system with embedded devices equipped with ARM CPUs and ARM Mali GPUs. Boards in this family have significant differences regarding the number of cores, memory size, frequencies, and GPU computational power. These differences remarkably impact the performance of multi-DNN workloads. It's important to clarify that the adaptability of FairBoost is a distinct advantage over the other state-of-art methods, which require re-training or fine-tuning to function optimally on different boards.

In our experiments, we used the Orange Pi 5 board as the initial training platform for the agent. We then transferred the agent's learned knowledge to the Odroid N2+ board. Both boards are equipped with a big.LITTLE CPU architecture and a Mali GPU. It's important to mention that the Odroid N2+ differs from the Orange Pi 5 in a few ways. For example, the Odroid N2+ has 2 LITTLE cores, whereas the Orange Pi 5 has 4. Additionally, the Odroid N2+ comes with 4 GB of RAM, in contrast to the 16 GB found in the Orange Pi 5. Despite these hardware differences, FairBoost was robust enough to be transferred to the Odroid N2+ without requiring any retraining or fine-tuning.

The robustness of our methodology is largely attributed to the incorporation of Reinforcement Learning (RL) as the foundational framework. Specifically, our lightweight ResNet-9 equips our workload manager with the robustness needed to handle changes in the dataset, which can be interpreted as "noise" injected by the variations between heterogeneous embedded devices [46]. Furthermore, the combination of the codebook's reduced size and the redundancy in the parameters of the VQ-VAE encoder further equips our framework with resilience against data drifts during platform switches. The synergy of these design elements renders our framework robust and enables knowledge transfer from one heterogeneous embedded system to another with minimal performance loss.

5 Experimental Evaluation

In this section, we demonstrate the strengths of FairBoost in terms of (**i**) VQ-VAE encoder training (Sec. 5.1); (**ii**) agent learning (Sec. 5.2); (**iii**) throughput/fairness comparison (Sec. 5.3); (**iv**) knowledge transfer (Sec. 5.4); and (**v**) runtime performance (Sec. 5.5). We evaluate several multi-DNN workloads on the Orange Pi 5 development board. The Orange Pi 5 features a Mali-G610 GPU and big.LITTLE CPUs with a quad-core Cortex-A76 at 2.4GHz and a quad-core Cortex-A55 at 1.8GHz. FairBoost is implemented in PyTorch [47] and OpenAI Gym [48]. Additionally, OpenCL and ARM Compute Library [45] are employed to develop the DNN pipelines on the embedded device.

5.1 VQ-VAE Encoder Training

Regarding the training of our VQ-VAE model, we performed DNN layer decomposition as elaborated in subsection 4.1. We considered 18 commonly utilized DNNs: (**i**) AlexNet, (**ii**) GoogleNet, (**iii**) Inception-ResNet V2, (**iv**) Inception V3, (**v**) Inception V4, (**vi**) LeNet, (**vii**) MobileNet, (**viii**) MobileNet V2, (**ix**) ResNet-12, (**x**) ResNet-50, (**xi**) ResNet-50 V2, (**xii**) ResNeXt-50, (**xiii**) ShuffleNet, (**xiv**) SqueezeNet, (**xv**) SSD with MobileNet backbone, (**xvi**) YOLO V3, (**xvii**) VGG-16, and; (**xviii**) VGG-19.

After extracting the raw layer representation, we generated our dataset, containing a total of $1,937$ samples. The dataset is then partitioned into training and test subsets with 0.9 split ratio. Given the limited number of samples in our dataset, we opted for K-Folds cross-validation with $K = 10$ to augment the dataset and consequently optimize the VQ-VAE performance. We selected Mean Squared Error (MSE)

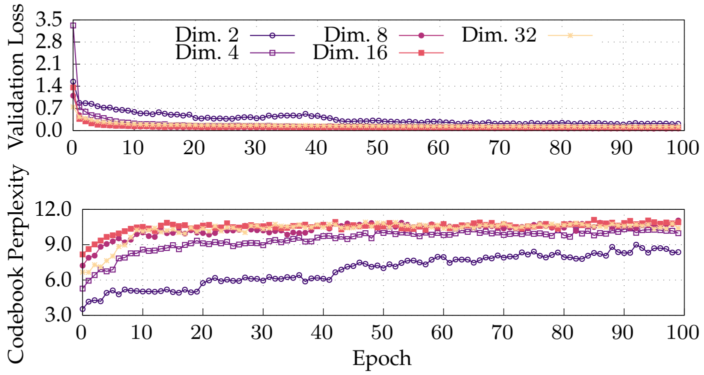


Fig. 6: VQ-VAE codebook validation loss and perplexity function latent dimension.

TABLE 2: Hyper-parameter list for our Rainbow-DQN agent.

Parameter	Value
Min history to start learning	$10K$ steps
Adam learning rate	1×10^{-3}
Exploration ϵ	0
Noisy Nets σ_0	0.5
Target Q-value estimator update period	500 steps
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	$[-21, 21]$
Gradient clipping by L_2 norm	5

as a criterion for the VQ-VAE training. To decide upon the codebook dimension, we performed evaluations of the VQ-VAE across various latent space dimensions. Specifically, we tested the dimensions of (**i**) 2; (**ii**) 4; (**iii**) 8; (**iv**) 16; and (**v**) 32. We chose powers of 2 for the latent space dimensions to ensure numerical stability over the output feature map dimensions of our Rainbow DQN agent's online model and streamline this hyperparameter's tuning. The results regarding validation loss and perplexity of the codebook for each latent dimension setting are depicted in Figure 6. We observe that the latent dimension balancing data encoding accuracy and computational efficiency is 8. The validation loss in a VQ-VAE module measures prediction error during evaluation, with lower values indicating improved performance. The codebook's perplexity curve depicts the diversity and distribution of embedding vectors, where a higher perplexity indicates a more efficient use of the codebook.

5.2 Agent Learning Strategy

The agent was trained on a machine equipped with an NVIDIA RTX 2070 GPU. In the initial stages of the training process, we employ the Kaiming initialization method to uniformly distribute the parameters of the online Q-value estimator [49]. This strategy effectively models the rectifiers of our custom ResNet-9 backbone, which powers the Rainbow-DQN agent, thereby contributing to the agent's rapid convergence. We build on the existing empirical work in the RL domain and maintain the number of stacked frames at 4. We also normalize the state matrix within $[0, 1]$. As Rainbow's components are associated with various hyper-parameters, hyper-parameter tuning is a significant combinatorial challenge. Given the enormity of the hyper-parameter space, conducting an exhaustive search is not feasible. A detailed breakdown of FairBoost' hyper-parameters can be found in Table 2.

Following the exploration phase comprising $10K$ steps, the agent continued training for an additional $20K$ steps. As we mentioned in Section 4, we significantly accelerate our agent's training process. This refers to the number of steps our agent experienced during training and does not directly represent time. Specifically, the training overall took approximately 10 days to complete. While standard RL frameworks often require training involving millions of steps, we dropped this required number of steps and accelerated our training by a factor of about $\times 1,000$. The performance of the agent is evaluated for both throughput and fairness. To that end, the environment is configured to interact directly with the Orange Pi 5 development board and execute the actions determined by the agent. Figure 7 shows the reward curve during the agent's training. We also show the training progress of an agent without action masking and caching, which demonstrates an undesirably unstable behavior from step to step, hitting both negative and positive rewards, and ultimately cannot converge. Each point holds 50 environment steps. Lines represent the minimum ("Min"), maximum ("Max"), and mean ("Average") rewards, offering insights into the reward fluctuations and overall agent's learning progress.

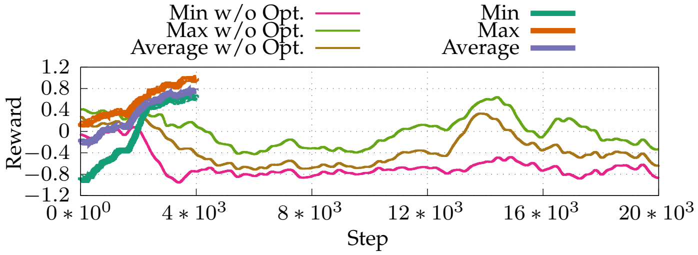


Fig. 7: Reward evolution over $20K$ steps, aggregated into $4K$ points. The training for the agent w/o any optimizations took about 50 days without converging.

5.3 Throughput and Fairness comparison

We deploy three key metrics to demonstrate the optimization in throughput and fairness. Table 3 provides the mathematical formulations associated with each one of them:

1) **Normalized Throughput**: The throughput of a multi-DNN workload, as defined in equation 1, over the baseline throughput. As baseline throughput, we define the achieved average throughput of all DNNs when all of them are executed on the GPU.

2) **Jain Fairness Index** ($\mathcal{J}$): The Jain Fairness Index [28] provides a quantitative measure to assess the fairness in performance among DNNs. It is defined in the range of $[0, 1]$. A value of 1 signals perfect fairness, indicating that the performance of all DNNs degraded equally in proportion relatively to their ideal throughput values.
3) **Overall Gain** ($\mathcal{O}$): This metric is devised to provide insights into the trade-off between throughput and fairness for each manager. A higher value indicates a superior balance between the two objectives.

TABLE 3: The evaluation metrics along with their formal definitions. N is the total number of DNNs in the workload. $t^i_{current}$ is inferences per second of DNN i when executed concurrently with the rest of the DNNs in the mix and t^i_{ideal} represents the throughput of DNN i when executed alone on the embedded GPU.

Metric	**Mathematical Definition**
Workload Throughput	$\mathcal{T}_w = \frac{1}{N} \times \sum_{i=1}^{N} t^i_{current}$
Normalized $\mathcal{T}$	$\frac{\mathcal{T}_w}{\mathcal{T}_{baseline}}$
Jain Fairness Index	$\mathcal{J} = \frac{\left(\sum_{i=1}^{N} \frac{t^i_{current}}{t^i_{ideal}}\right)^2}{N \cdot \sum_{i=1}^{N} \left(\frac{t^i_{current}}{t^i_{ideal}}\right)^2}$
Overall Gain	$\mathcal{O} = \mathcal{T} \times \mathcal{J}$

We compared our framework against six other approaches: (**i**) the common approach, which maps all DNNs on the GPU, the highest-performing computing component (also used as the baseline); (**ii**) a linear regression-based algorithm adopted by MOSAIC [37]; (**iii**) the ODMDEF [38], utilizing both a linear regression-based method and a k-NN classifier as detailed in their experimental assessment; (**iv**) the Genetic Algorithm (GA) presented in [2]; (**v**) a round-robin variation that cycles through DNNs and statically assigns them on the big CPU cluster, named RR (CPU); and (**vi**) a round-robin variation that cycles through DNNs and statically assigns them on the GPU, named RR (GPU). We included Round-robin scheduling to expand our analysis. Round-robin scheduling assigns an equal, fixed amount of processing time to each DNN in the workload. The scheduler then cycles through these DNNs, providing each with its allocated time quantum before saving its state and proceeding to the next. This method is particularly challenging for complex and long-running DNNs, as it requires efficient state management to allow each DNN to resume processing seamlessly. The rationale behind a CPU variation of round-robin and a GPU one was the difference regarding memory overhead between the two instances.

Figure 8 depicts the memory overhead regarding DNNs mapped on either the CPU or GPU and how many seconds were left for them for inference out of 30 seconds. Regarding DNNs mapped on the GPU, we observe that most of the time was used to load the weights and data on the memory, thus introducing memory overhead. On average, when a model was entirely loaded on the CPU, the memory overhead was 2 seconds. This overhead is ×9.5 higher in the case of GPU. To that end, we studied a round-robin variation that utilizes only the CPU (both the big and LITTLE clusters) and a second one that utilizes only the GPU. To perform an in-depth study, we created random mixes of multi-DNN workloads. Specifically, we considered mixes of 3, 4, and 5 concurrent DNNs. We also attempted to evaluate mixes of 6 concurrent DNNs, but the workload exerted an overwhelming strain on both the memory controller and the computational resources, rendering the board unresponsive. We utilized the Orange Pi 5 board, which features a Mali-G610 GPU and big.LITTLE CPUs with a quad-core Cortex-A76 running at 2.4GHz and a quad-core Cortex-A55 at 1.8GHz.

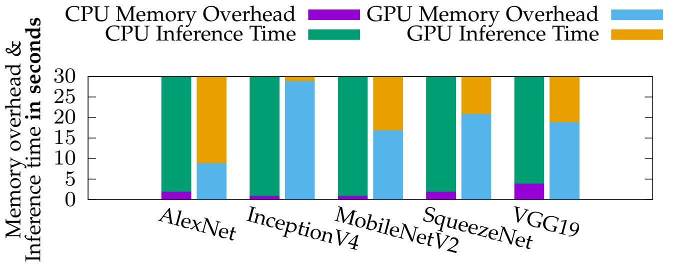


Fig. 8: Memory overhead over Inference time ratio for DNNs mapped entirely either on CPU or GPU.

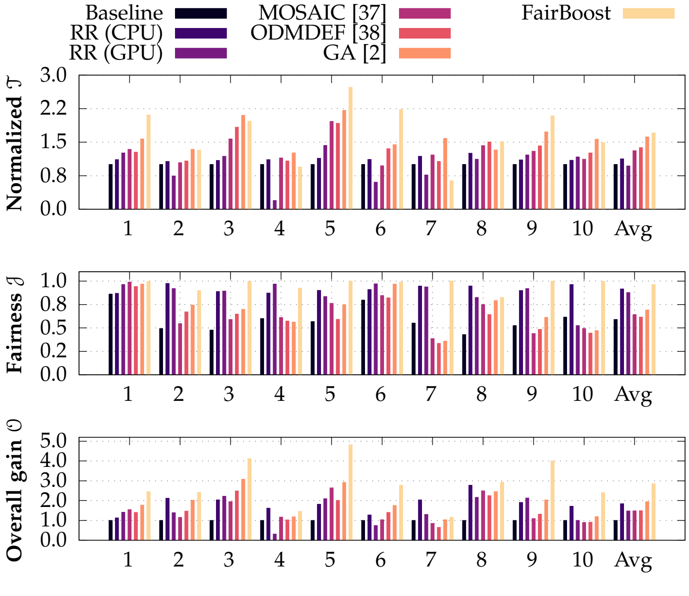


Fig. 9: Comparison of (a) Normalized Throughput $\mathcal{T}$; (b) Jain fairness index $\mathcal{J}$; and (c) Overall throughput/fairness gain $\mathcal{O}$ for ten different mixes. Each mix consists of 3 concurrent executing DNNs, selected randomly.

Mixes of 3 DNNs: Figure 9 depicts the experimental results for the case of ten random workload mixes, each one consisting of 3 concurrent DNNs. In Figure 9(c), we observe that FairBoost improves on average the overall gain ($\mathcal{O}$) by ×2.85, 54.8%, 93.2%, ×0.92, ×0.91, and ×0.47 compared to the baseline, RR (CPU), RR (GPU), MOSAIC, ODMDEF, and the GA, respectively. This enhancement can be further analyzed for throughput and fairness. To that end, Fair-

Boost surpasses the GA in throughput by 5.6% on average, as shown in Figure 9(a). Meanwhile, the GA significantly compromises the fairness metric, which becomes evident in mixes 7 and 10, where GA shows a preference for a single DNN. This negatively impacts fairness, as shown in Figure 9(b), demonstrating the inherent drawback of GA in balancing fairness against throughput. In contrast, FairBoost delivers a superior fairness performance across all mixes, leading to a greater overall gain. This is attributed to FairBoost's balanced incorporation of both throughput and fairness, while all the other frameworks emphasize throughput over fairness. Specifically, FairBoost achieves 96.2% Jain fairness score, while the baseline, RR (CPU), RR (GPU), the MOSAIC, the ODMDEF, and the GA, scored 59%, 91.6%, 87.4%, 64%, 61.4%, and 69% respectively in that category. While round-robin scheduling provides equal time to each DNN (fair in terms of execution time), it does not lead to efficient throughput. The context-switching overhead, crucial for saving and loading states of DNNs, is particularly burdensome for larger models, thus reducing the system's overall computational throughput. Therefore, FairBoost efficiency distributes the available resources even in complex scenarios involving multiple DNNs, particularly in lighter workloads where optimal partition points are easier to identify. It's noteworthy that running 3 concurrent DNNs doesn't saturate the board's computational resources, allowing for comparable throughput solutions across frameworks. This is more apparent in mixes 4 and 8, constituted mainly of lightweight DNNs like ShuffleNet, ResNet-12, SqueezeNet, AlexNet, and GoogleNet.

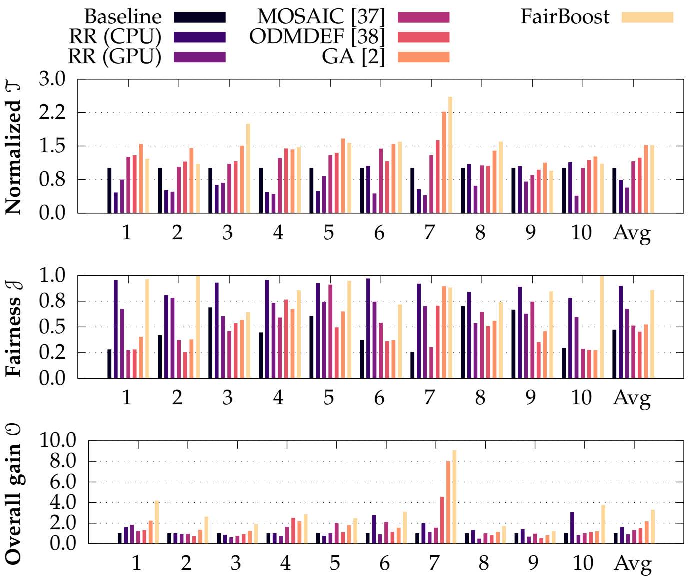


Fig. 10: Comparison of (a) Normalized Throughput $\mathcal{T}$; (b) Jain fairness index $\mathcal{J}$; and (c) Overall throughput/fairness gain $\mathcal{O}$ for ten different mixes. Each mix consists of 4 concurrent executing DNNs, selected randomly.

Mixes of 4 DNNs: We further tested the robustness of FairBoost under increased workload with mixes of 4 concurrent DNNs, as shown in Figure 10. Despite the increased workload, FairBoost consistently outperformed the baseline, RR (CPU), RR (GPU), MOSAIC, ODMDEF, and the GA in terms of overall throughput/fairness gain, demonstrating ×3.26, ×2.1, ×3.7, ×2.52, ×2.26, and ×0.53 better $\mathcal{O}$ respectively, as illustrated in Figure 10(c). This underscores the limitations of the baseline, MOSAIC, and ODMDEF methods, which led to excessive GPU utilization. Furthermore, regarding RR (CPU) and RR (GPU), it becomes more apparent that the underlying heterogeneity is not fully leveraged, resulting in an overall average gain that is up to ×3.7 worse. Both GA and FairBoost, in contrast, exhibited better utilization of the platform's computing components and improved system throughput $\mathcal{T}$. Furthermore, FairBoost ties with the GA in terms of throughput, as shown in Figure 10(a). Figure 10(b) reveals the shortcomings of GA in mixes 1, 2, and 10, where it unfairly manages one DNN, adversely affecting the others. FairBoost maintained fair throughput distribution among DNNs, which is translated in 81.8%, 67.6%, 89.8%, and 64.3% better $\mathcal{J}$ fairness compared to the baseline, MOSAIC, ODMDEF, and the GA, respectively. RR (CPU) demonstrates 5% better Jain Index than FairBoost, while RR (GPU) is 27.3% less fair than FairBoost. The difference in fairness between the two RR instances is attributed to context switching. Specifically, for the GPU instance of RR, there is a significant memory overhead, leading to poor multi-DNN mappings. The potent advantage of FairBoost lies in its ability to comprehend the intricate relationships among various factors involved in efficient resource utilization, as opposed to linear regression-based frameworks, such as MOSAIC and ODMDEF. A noticeable exception occurs in mix 7, where the GA achieves better fairness than FairBoost. However, FairBoost retains a higher overall throughput/fairness gain, as shown in Figure 10(c). Interestingly, all compared approaches exhibit improved fairness over the case of mixes of 3 concurrent DNNs. This improvement is attributed to the increased complexity of the workload, which necessitates more resource allocation and, consequently, reduces individual DNN throughput. In other words, the state-of-the-art approaches yield pipelines that saturate the platform's resources, leading to a uniformly decreased DNN throughput.

Mixes of 5 DNNs: Advancing to the more demanding scenario of 5 concurrent DNNs, the limitations of the embedded device become clear. Even though ODMDEF, GA, and FairBoost effectively used the system's varied hardware components, managing the increasingly demanding workloads became significantly more difficult. The simultaneous execution of five DNNs strained the computational resources, culminating in throughput saturation. This is evident in Figure 11(a), where all frameworks exhibit similar behavior in terms of throughput. Despite these constraints, Figure 11(c) shows that FairBoost exhibited impressive resilience with its deep reinforcement learning approach, outperforming the other methodologies in overall throughput/fairness gain. Specifically, FairBoost outperformed the baseline, RR (CPU), RR (GPU), MOSAIC, ODMDEF, and GA by factors of ×3.61, ×1.99, ×7.27, ×2.08, ×0.99, and ×0.55 in $\mathcal{O}$, respectively. Moreover, under such demanding circumstances, FairBoost managed to yield relatively fair multi-DNN pipelines. This was manifested in a slightly reduced throughput $\mathcal{T}$, but significantly enhanced fairness $\mathcal{J}$, as shown in mixes 1 and 10 in Figure 11(b). In these mixes,

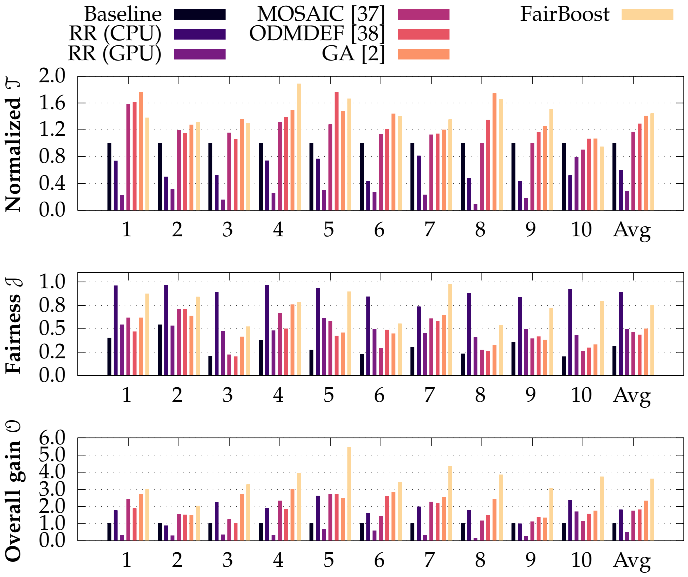


Fig. 11: Comparison of (a) Normalized Throughput $\mathcal{T}$; (b) Jain fairness index $\mathcal{J}$; and (c) Overall throughput/fairness gain $\mathcal{O}$ for ten different mixes. Each mix consists of 5 concurrent executing DNNs, selected randomly.

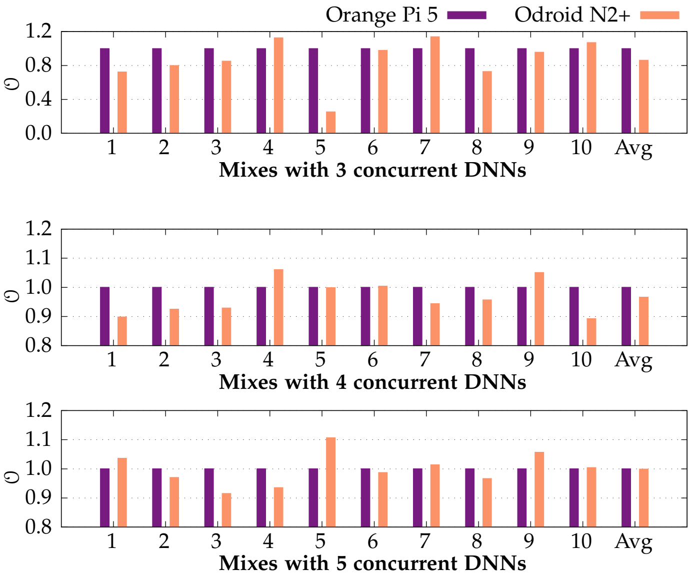


Fig. 12: Comparison of normalized overall gain $\mathcal{O}$, after transferring the agent from Orange Pi 5 (source platform) to the Odroid N2+ (target platform) without retraining or fine-tuning considering (a) 3 concurrent DNNs; (b) 4 concurrent DNNs; and (c) 5 concurrent DNNs;

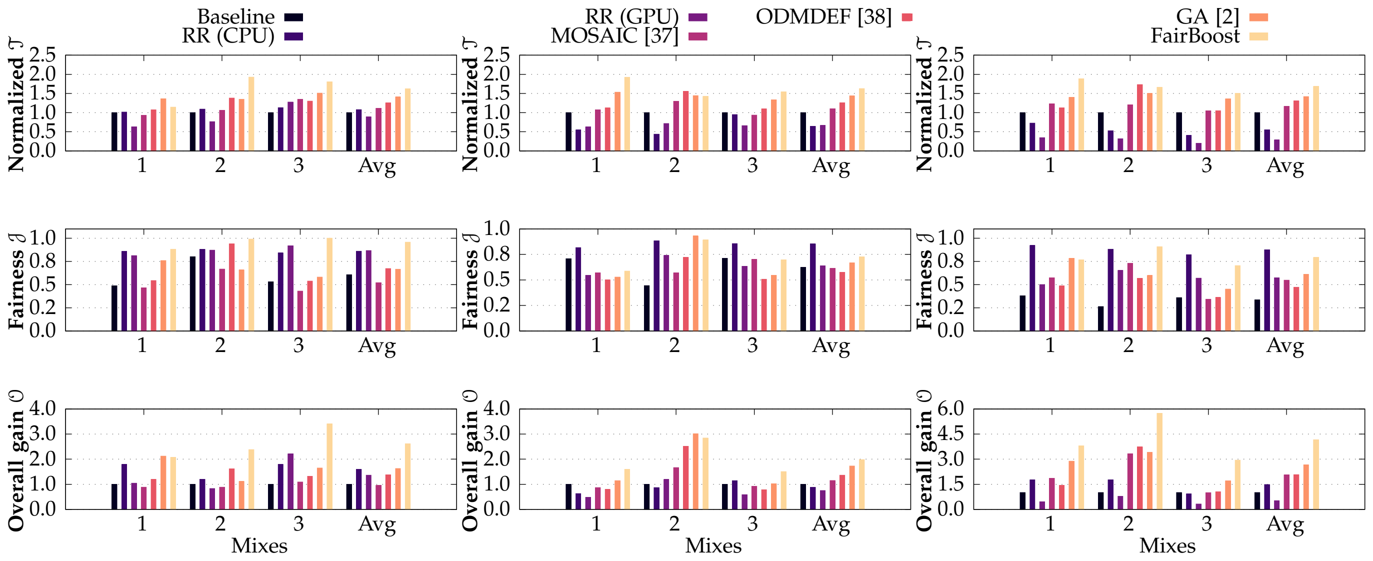
the GA preferentially managed a single DNN, boosting the system throughput at the cost of a significant decline in fairness. FairBoost also achieved superior fairness compared to the baseline, MOSAIC, ODMDEF, and GA, showing improvements by factors of $\times 2.41$, $\times 0.63$, $\times 0.73$, and $\times 0.5$ respectively, as shown in Figure 11(b). Compared to RR (CPU), FairBoost achieved 26% lower Jain Fairness index. However, it is evident from Figure 11(a) that RR(CPU) starves most of the DNNs from resources, ultimately leading to $\times 2.44$ worse average normalized throughput $\mathcal{T}$. These results demonstrate FairBoost's properties in managing highly complex, multi-DNN workloads. Despite these highly demanding workloads, FairBoost managed to make intelligent decisions and manage the workload to yield better overall throughput and fairness.

5.4 Knowledge transfer evaluation

Our framework's capability to transfer knowledge was evaluated by porting the pre-trained agent from Orange Pi 5 board to the Odroid N2+ development board. The transferred agent was subjected to the same mixes as detailed in Section 5.3. This approach allowed us to directly compare the performance of the transferred agent on the target platform, with no additional retraining or fine-tuning against the source platform, thus testing the robustness of our system under diverse conditions.

Figure 12 depicts the performance of the transferred agent in terms of overall throughput/fairness gain. Remarkably, the transferred agent demonstrated a slight average performance reduction of only 5.7% compared to the agent trained on the source platform. The effectiveness of our method is largely due to the VQ-VAE module, which excels at identifying underlying data patterns. This enables the framework to smoothly adapt to any data variations that might occur when moving it to a new platform. The Rainbow-DQN agent also plays a crucial role by leveraging NoisyNets to explore its environment, thereby adapting to scenarios with minor variations in data. Importantly, the transferred agent is efficient in terms of resource use, saving considerable computational power and time that would have been spent on additional training. This result highlights the effectiveness of our knowledge transfer method, establishing it as a strong and practical solution for handling complicated workloads involving multiple DNNs across diverse heterogeneous embedded systems.

To further demonstrate the effectiveness of knowledge transfer we have performed additional experiments on the Odroid N2+ board. These experiments are particularly noteworthy because: (**i**) we conducted a direct comparison of throughput and fairness metrics across different scenarios. (**ii**) the workloads used were new to the estimator and had not been previously evaluated on the Orange Pi 5. (**iii**) our method was applied without any re-training or fine-tuning. Through these steps, we aim to provide a clearer and more robust evaluation of knowledge transfer, thereby addressing the concerns raised about the suitability of our chosen metric. We created 9 previously unseen multi-DNN mixes, each involving different numbers of concurrently executed DNNs. Specifically, we created 3 mixes for 3 concurrent DNNs, another 3 mixes for 4 concurrent DNNs, and 3 more mixes for 5 concurrent DNNs. Figure 13 demonstrates the advantage of FairBoost on overall performance gain of $\times 2.62$ over the baseline and $\times 1.61$ over the GA for mixes of 3 concurrently executing DNNs. FairBoost maintains this advantage in the case of 4 DNNs. Specifically, it has a 98% and 15% higher overall gain compared to the baseline and GA, respectively. We observe a small drop in average overall



(a) Mixes of 3 concurrent DNNs

(b) Mixes of 4 concurrent DNNs

(c) Mixes of 5 concurrent DNNs

Fig. 13: Comparison of Normalized Throughput $\mathcal{T}$; Jain fairness index $\mathcal{J}$; and Overall throughput/fairness gain $\mathcal{O}$ for 3 different mixes, each of (a) 3, (b) 4, and (c) 5 randomly selected DNNs.

gains compared to mixes of 3 DNNs, mostly because of the second mix where the GA achieves unusually better management. At this point, we understand that the gap could be alleviated if FairBoost was retrained or fine-tuned. However, since this is a single case and not a repeating pattern, we can safely consider this mix an outlier. Finally, regarding mixes of 5 DNNs FairBoost surpasses by a factor of $\times 4.15$ and $\times 1.56$ the baseline and the GA, respectively.

In conclusion, these experimental results not only emphasize the utility and adaptability of our approach but also signal its potential for a broader impact, providing a robust solution for handling complex multi-DNN workloads across an expansive range of heterogeneous embedded systems.

5.5 Run-time performance evaluation

We evaluated the time required for each framework to process a random mix of multiple DNNs, specifically with 3, 4, and 5 DNNs running concurrently. As expected, the baseline method was the fastest, as it simply allocated all DNNs to the GPU without any decision-making overhead. However, this approach failed to take full advantage of the system's diverse hardware, resulting in lower overall gain $\mathcal{O}$, as shown in Figures 9(c)-11(c). In contrast, both MOSAIC and ODMDEF had relatively quick inference times of approximately ~ 1 second. But they were slowed down by the extensive data collection needed for their decision-making processes. MOSAIC, for instance, needed over $35K$ data points for just one query to its trained linear regression model. ODMDEF was even more data-intensive, requiring two queries per DNN layer to the same model and another query to a trained k-NN classifier, totaling over $250K$ data points. Despite the speed of decision-making, both frameworks yield sub-optimal results, compromising both throughput and fairness, as shown in Figures 9(c)-11(c). The GA method, requiring retraining for each workload and an optimization layer to reduce pipeline stages, extends run-time response despite optimization in overall gain $\mathcal{O}$. This method took ~ 5 minutes for each mix in our tests.

Notably, FairBoost, manages to maintain satisfactory run-time performance due to the low number of trainable parameters. It determined an efficient mapping in ~ 2 seconds, demonstrating the best balance between run-time performance and overall throughput/fairness gain $\mathcal{O}$, rendering the most efficient framework.

6 Limitations

FairBoost targets both fairness and throughput in multi-DNN workloads on heterogeneous embedded systems. Still, we are planning on more extensions. Specifically, the existing managers, including FairBoost, do not deploy any DNN compression mechanism, and thus, they cannot support higher-order workloads. To that end, we plan on extending FairBoost with a synergistic pruning and quantization module that can further expand the capabilities of modern embedded devices regarding both workload requirements and QoS. Furthermore, while it facilitates knowledge transfer between platforms, FairBoost is limited to embedded systems that support ARM CL. To that end, we are planning on extending the backend of our framework and supporting NVIDIA platforms, such as the Jetson Orin. This could open new challenges in terms of knowledge transfer, and we would most likely have to reformulate the input of the agent. Specifically, the agent would have to become multi-modal and accept information on the complexity of each DNN in the workload as well as the features of the different computing components. Overall, we are planning on exploring several additional aspects that come with resource-constrained environments in embedded systems.

7 Conclusion

The rise of Deep Neural Networks (DNNs) has resulted in complex workloads employing multiple DNNs concurrently. This trend raises challenges in managing these multi-DNN workloads efficiently, especially in heterogeneous embedded systems. Current workload managers cannot efficiently handle such workloads leading to reduced system throughput, mainly due to contention and uneven performance among various DNN models. In this work, we propose FairBoost, a run-time manager targeting both system throughput and fair multi-DNN execution. Our evaluation highlights the efficiency of FairBoost in managing multi-DNN workloads. In our experiments involving 18 DNNs across a range of scenarios, FairBoost demonstrates a significant average improvement in throughput and fairness by $\times 3.24$ when compared to the state-of-art. Moreover, FairBoost demonstrates robustness in knowledge transfer allowing us to transfer the agent across multiple boards without any retraining or fine-tuning. In summary, FairBoost addresses the complexities of managing multi-DNN workloads on heterogeneous embedded systems and can efficiently produce pipelines that co-optimize throughput and fairness.

Acknowledgment

This work is supported in part by grant NSF CCF 2324854.

References

[1] 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.

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

[3] 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.

[4] B. Cox *et al.*, "Masa: Responsive multi-dnn inference on the edge," in *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2021, pp. 1–10.

[5] H. Kwon, L. Lai, T. Krishna, and V. Chandra, "Herald: Optimizing heterogeneous dnn accelerators for edge devices," *arXiv preprint arXiv:1909.07437*, vol. 57, 2019.

[6] 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.

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

[8] A. Karatzas and I. Anagnostopoulos, "Omniboost: Boosting throughput of heterogeneous embedded devices under multi-dnn workload," *arXiv preprint arXiv:2307.03290*, 2023.

[9] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE micro*, vol. 28, no. 3, pp. 42–53, 2008.

[10] S. Kim, J. Zhao, K. Asanovic, B. Nikolic, and Y. S. Shao, "Aurora: Virtualized accelerator orchestration for multi-tenant workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 62–76.

[11] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, "Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 828–841.

[12] H. Fan, S. I. Venieris, A. Kouris, and N. Lane, "Sparse-dysta: Sparsity-aware dynamic and static scheduling for sparse multi-dnn workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 353–366.

[13] Y. Li, A. Louri, and A. Karanth, "A high-performance and energy-efficient photonic architecture for multi-dnn acceleration," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[14] S. I. Venieris, C.-S. Bouganis, and N. D. Lane, "Multiple-deep neural network accelerators for next-generation artificial intelligence systems," *Computer*, vol. 56, no. 3, pp. 70–79, 2023.

[15] 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.

[16] 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.

[17] W. Jiang *et al.*, "Exploiting potential of deep neural networks by layer-wise fine-grained parallelism," *Future Generation Computer Systems*, vol. 102, pp. 210–221, 2020.

[18] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[19] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[20] X. Liu, S. Li, M. Kan, J. Zhang, S. Wu, W. Liu, H. Han, S. Shan, and X. Chen, "Agenet: Deeply learned regressor and classifier for robust apparent age estimation," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2015, pp. 16–24.

[21] G. Levi and T. Hassner, "Age and gender classification using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 34–42.

[22] Techcrunch, "Qualcomm next-gen xr chip promises up to 4.3k resolution per eye," https://techcrunch.com/2024/01/04/qualcomm-next-gen-xr-chip-promises-up-to-4-3k-resolution-per-eye/.

[23] O. Pi. (2022) Orange pi 5. [Online]. [Online]. Available: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html

[24] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, 2017.

[25] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[26] K. He *et al.*, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[27] X. Zhang *et al.*, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[28] R. Jain, A. Durresi, and G. Babic, "Throughput fairness index: An explanation," in *ATM Forum contribution*, vol. 99, no. 45, 1999.

[29] C.-Y. Hsieh *et al.*, "The case for exploiting underutilized resources in heterogeneous mobile architectures," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019.

[30] M. Alzantot *et al.*, "Rstensorflow: Gpu enabled tensorflow for deep learning on commodity android devices," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*, 2017, pp. 7–12.

[31] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[32] S. S. Latifi Oskouei *et al.*, "Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 1201–1205.

[33] Y. Su *et al.*, "Joint dnn partition and resource allocation optimization for energy-constrained hierarchical edge-cloud systems," *IEEE Transactions on Vehicular Technology*, 2022.

[34] E. Baek *et al.*, "A multi-neural network acceleration architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 940–953.

[35] 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, pp. 235–247.

[36] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 392–405.

[37] 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.

[38] 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.

[39] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[40] D. Hafner *et al.*, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[41] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[42] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[43] C. Berner *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[44] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *arXiv preprint arXiv:2006.14171*, 2020.

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

[46] J. Djolonga *et al.*, "On robustness and transferability of convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 458–16 468.

[47] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[48] G. Brockman *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[49] K. He *et al.*, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

Andreas Karatzas received the Integrated Master degree (Diploma) from the department of Computer Engineering and Informatics (CEID), University of Patras, Patras, Greece, in 2021. He is currently pursuing the Ph.D. degree at the School of Electrical, Computer and Biomedical Engineering at Southern Illinois University, Carbondale, Illinois, as a member of the Embedded Systems Software Lab. His research interests include embedded systems, approximate computing, and deep learning.

Iraklis Anagnostopoulos is an Associate Professor at the School of Electrical, Computer and Biomedical Engineering at Southern Illinois University, Carbondale. He is the director of the Embedded Systems Software Lab, which works on run-time resource management of modern and heterogeneous embedded many-core architectures, and he is also affiliated with the Center for Embedded Systems. He received his Ph.D. in the Microprocessors and Digital Systems Laboratory of National Technical University of Athens. His research interests lie in the area of approximate computing, heterogeneous hardware accelerators, and hardware/software co-design.