Adaptive Workload Distribution for Accuracy-aware DNN Inference on Collaborative Edge Platforms

Zain Taufique
University of Turku
zatauf@utu.fi

Antonio Miele
Politecnico di Milano
antonio.miele@polimi.it

Pasi Liljeberg
University of Turku
pasi.liljeberg@utu.fi

Anil Kanduri
University of Turku
spakan@utu.fi

***Abstract*—DNN inference can be accelerated by distributing the workload among a cluster of collaborative edge nodes. Heterogeneity among edge devices and accuracy-performance trade-offs of DNN models present a complex exploration space while catering to the inference performance requirements. In this work, we propose adaptive workload distribution for DNN inference, jointly considering node-level heterogeneity of edge devices, and application-specific accuracy and performance requirements. Our proposed approach combinatorially optimizes heterogeneity-aware workload partitioning and dynamic accuracy configuration of DNN models to ensure performance and accuracy guarantees. We tested our approach on an edge cluster of Odroid XU4, Raspberry Pi4, and Jetson Nano boards and achieved an average gain of 41.52% in performance and 5.2% in output accuracy as compared to state-of-the-art workload distribution strategies.**

***Index Terms*—Distributed Edge; Approximation; DNN Inference;**

I. Introduction

Deep Neural Networks (DNNs) provide smart and autonomous services in application domains such as Internet of Things (IoT), by processing continuous input data streams from multiple sources. The compute-intensive DNN inference requests are typically offloaded to resourceful cloud servers to handle the compute requirements. However, continuous data transfer between user-end edge devices and remote cloud servers affects quality-of-service with increased inference latency, disconnection risks, and data privacy concerns [1]. To address these challenges, DNN inference requests can be distributed across a cluster of locally connected edge devices for collaborative execution [2]. Edge clusters are increasingly becoming heterogeneous, where the connected devices have different hardware architectures and computational capabilities [3], [4]. On the other hand, widely used pruned versions of DNN models present different accuracy-performance trade-offs [5]. Combining the node-level heterogeneity of edge devices and accuracy-performance trade-offs of different DNN models exposes a significantly wider Pareto space to navigate through for efficient workload distribution. We present the performance diversity among a heterogeneous cluster of edge devices viz., Odroid XU4, Raspberry Pi4, and Jetson Nano boards, while running image classification on pre-trained MobileNetV2 [6]. We executed 6 different versions of MobileNet models (a0-a5), exhibiting varying levels of accuracy (92.5% - 82.9%) with pruned model parameters by reducing the multiplication width [7]. Figure 1 shows the performance and

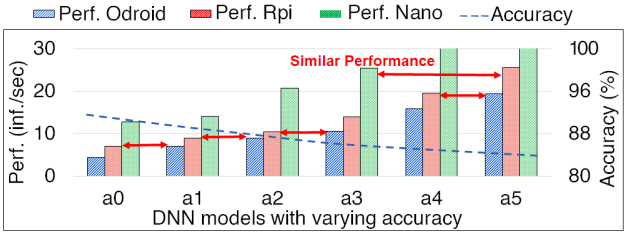

Figure 1: Accuracy-perfomance trade-offs of inferring MobileNetV2 across different edge devices.

accuracy of different MobileNet models (a0-a5) across heterogeneous edge devices. While each edge device has specific inherent compute capabilities, different devices can still deliver a similar performance (inferences-per-second) by tweaking the accuracy of the DNN model. For example, Jetson Nano provides better performance than Odroid XU4 and Raspberry Pi4 across different models. However, both Raspberry Pi4 and Odroid XU4 can provide a similar performance to that of Jetson Nano (shown in red arrows in Figure 1), although at a lower accuracy. This demonstrates a non-intuitive accuracy-performance-heterogeneity optimal space that can be exploited for efficient workload distribution.

Existing strategies for distributing DNN inference workloads across edge clusters use input data partitioning [2], [8] and/or DNN model splitting [3], [9]–[11], depending on application-specific scenarios. However, these strategies consider (i) accuracy-performance trade-offs assuming homogeneous edge nodes, or (ii) heterogeneous edge nodes ignoring accuracy-performance trade-offs. Consequently, these approaches have limited efficacy in handling dynamic scenarios such as run-time workload variation and device availability. Addressing these challenges necessitates an intelligent workload distribution policy that combinatorially optimizes workload partitioning and accuracy configuration– to meet performance and accuracy constraints, considering heterogeneity and availability of edge nodes.

In this work, we present an adaptive workload distribution strategy to dynamically partition, distribute, and set the accuracy levels of DNN inference workloads executed over collaborative heterogeneous edge clusters. We design an edge cluster framework that supports workload distribution among connected heterogeneous edge nodes by monitoring run-time system and workload dynamics. Our workload distribution policy makes online decisions on workload partitioning, distri-

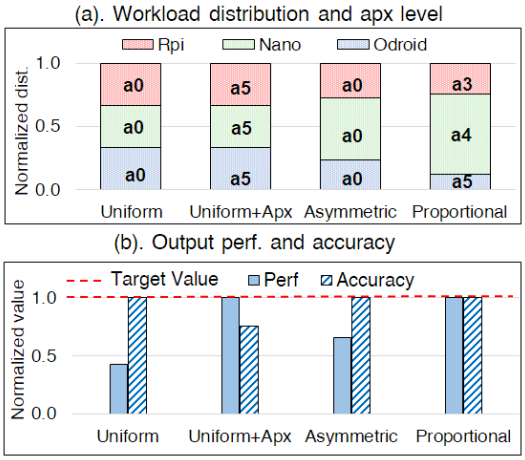

Figure 2: Workload distribution through different strategies. (a) workload distribution and selected approximation level on each board. (b). Overall output performance and accuracy.

bution, and accuracy configuration of DNN inference requests, optimizing the collaborative resource utilization across the edge cluster, while meeting the performance and accuracy requirements. Our proposed policy sets the accuracy levels of DNN models through a dynamic selection of appropriate DNN kernels from a pool of pre-trained models that exhibit different performance-accuracy trade-offs [5]. Our contributions are: (i) design of edge cluster framework supporting run-time monitoring of system dynamics and workload distribution, (ii) heterogeneity aware adaptive workload distribution policy exploiting accuracy-performance trade-offs (iii) deployment of our proposed framework on real hardware edge cluster (Odroid XU4, Raspberry Pi4, and Jetson Nano) and evaluation against state-of-the-art workload distribution strategies.

II. Motivation and Related Work

A. Motivation

Traditionally, DNN workloads are distributed in an edge cluster: (i) *uniformly* – equally distributing the workload across different devices [10], [11], (ii) *asymmetrically* – distributing the workload proportional to the devices' compute capabilities [3], [9], [12], or (iii) *uniformly with approximation* – evenly distributing the workload and aggressively approximating to meet performance requirements [5]. Distributing the workload uniformly in heterogeneous environments causes an unbalanced use of resources that leads to performance violations in harsh workload scenarios. Asymmetrically distributing the workload according to node capability provides relatively better performance by optimizing the usage of the available resources. However, the lack of approximation limits this strategy's performance to the rated capacity of the available nodes. Finally, the uniform distribution with approximation can achieve the required performance of intense workloads but aggressively sacrificing the accuracy may violate the accuracy requirements of the workload. Given the limitations of the aforementioned approaches, a *Proportional* workload distribution strategy is required to determine optimal partitioning that meets the performance requirements within the minimum possible approximation level. We demonstrate the distribution of a compute-intensive DNN inference workload using the *uniform, uniform+apx, asymmetric* and *proportional* approaches in an edge cluster of Raspberry Pi4, Jetson Nano, and Odroid XU4 boards as shown in Figure 2 (a). For each strategy, we show the stacked percentile of workload distributed along with the selected DNN model across different devices. The block size within the stack shows the workload partition size, the color of the block represents the device, and the model number (a0-a5) represents the approximation level of the selected model. We present the output accuracy and performance of all these strategies in Figure 2 (b). The *Uniform* and *Uniform+apx* strategies distribute the workload equally among the available boards. The *Uniform* approach cannot achieve the performance but is successfully meeting the accuracy requirements because it does not approximate the workload. The *Uniform+apx* is utilizing approximation to meet the performance requirements, but its aggressive use of approximation violates the accuracy requirements of the workload. The workload is distributed unevenly in both *Asymmetric* and *Proportional* approaches. However, the *Asymmetric* approach cannot meet the required performance because the workload requirements are far higher than the limitation of the available devices. The *Proportional* strategy is heterogeneity-aware and achieves the target accuracy and performance by intelligently partitioning and approximating the workload proportionally to each board's computational capacity. With the insights from the demonstrative example in Figure 2, we posit that (i) workload partitioning and accuracy configuration should be fine-tuned together to meet performance and accuracy requirements, and (ii) edge node-level heterogeneity-awareness is the key for jointly actuating workload partitioning and approximation. Our goal is to design an adaptive workload distribution policy that outlines the *Proportional* strategy to ensure optimal usage of the edge cluster while meeting the DNN application requirements.

B. Related Work

Run-time workload distribution of DNN applications to achieve the target performance and accuracy while optimizing the usage of heterogeneous resources is a complex challenge. Most research works [10], [11] evenly partition the workload and are suitable for homogeneous edge clusters. In contrast, recent works [1], [3], [9], [12], [13] consider heterogeneity for workload distribution with design-time knowledge of workload conditions and the available resources. Zhou et al. [12] and Legion [3] presented adaptive frameworks that partition the workload based on the device performance and the network

Table I: Comparison of our and existing approaches.

	[1]	[9]	[10]	[11]	[3]	[5]	[12]	[13]	**Our**
Perf. aware	✓	✓	✓	✓	✓	✓	✓	✓	✓
Acc. aware	✓	✘	✘	✓	✓	✓	✘	✘	✓
Heter. aware	✓	✓	✘	✘	✓	✘	✓	✓	✓
Adaptive	✓	✘	✘	✓	✓	✓	✓	✓	✓
Run-time	✘	✘	✘	✘	✘	✓	✘	✘	✓

✓: supported, ✘: not supported

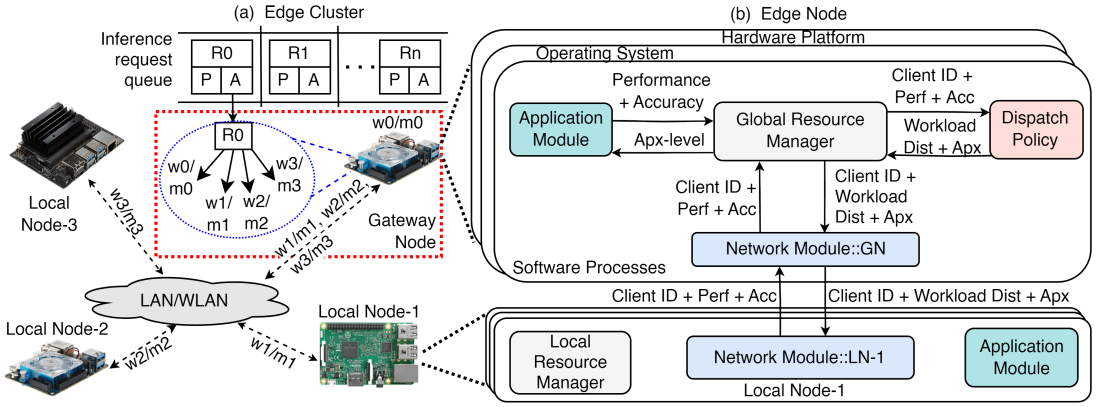

Figure 3: System Diagram showing (a) edge cluster (b) the node-level software and hardware modules

bandwidth but suffer if a device shuts down during the execution unexpectedly. AutoDice [1] extended these solutions to form an autonomous framework that performs scenario-based workload partitioning with performance awareness, given that the user defines the available resources at design time, and these resources remain available during the entire execution cycle. Sina et al. [5] considered the availability of the resources and presented an accuracy-aware workload distribution strategy for the edge-cloud paradigm, but the solution considers resource homogeneity while distributing the workload. We propose an advanced workload distribution framework that exceeds the state-of-the-art approaches by having performance, accuracy, and heterogeneity awareness and adapting to varying application and resource scenarios, as shown in Table I.

III. Adaptive Workload Distribution

A. Framework Overview

We have designed an edge cluster framework comprising heterogeneous hardware nodes connected with wLAN, as shown in Figure 3 (a). Each node within the cluster handles incoming applications, network connection, workload distribution, and resource allocation through Operating System (OS) level support. Figure 3 (b) presents an overview of the designed software infrastructure in an edge node. We have considered a hierarchical architecture for the edge cluster, including a leader Gateway Node (GN) and multiple followers Local Node (LN). The GN receives a queue of inference requests (R_0,R_1,...,R_n) where each request has a batch of images with specific performance and accuracy requirements ($P|A$). We assume that the inference requests can be serviced by selecting a model from a set of models exhibiting variable accuracy-performance trade-offs. The GN partitions the workload in a data-parallel manner (w_0,w_1,...,w_n) and selects the appropriate model (m_0,m_1,...,m_n), enabling the LNs to meet the required performance and accuracy constraints. We have generalized our design to allow any node to be a GN as per user requirements. The platform design for intelligent workload distribution is discussed in the following.

Hardware Platform and OS. The distributed hardware platforms may comprise multiple heterogeneous multi-core nodes. Each node supports run-time power monitoring through onboard power sensors or external monitoring equipment. We considered the edge node platforms with Dynamic Voltage/Frequency Scaling (DVFS) actuation support to restrict the power consumption of the board below a maximum power consumption limit called Thermal Design Power (TDP), for thermal safety. Each edge node runs a Linux-based OS to enable interaction between the node hardware and software modules. We use the OS interfaces at run-time to map the workload to the CPU cores, monitor CPU utilization and power consumption, and actuate DVFS to maximize performance within fixed TDP. The *Application Module* loads the required libraries and kernels to perform the inference of the selected model within a given workload; and stores the resultant performance and accuracy measures. The hardware nodes require a network connection with ethernet or WiFi support to enable inter-board communication. The *Network Module* enables the GN to exchange data with a dedicated LN at run-time and provides a list of available devices.

Workloads. We have designed the framework to process DNN inference workload with defined performance and accuracy requirements. We consider inference workload for streaming applications where the input data can be partitioned into smaller batches and distributed among the available nodes for parallel inference. We have considered scenario-based applications that have dynamic inference requests of variable input sizes, performance, and accuracy requirements. A common example of our considered applications is smart video surveillance [2], where the inference request data is a set of image frames that can be divided into small batches and distributed to the edge cluster for parallel inference. In our designed framework, each node has saved pre-trained models that can be used at run-time to provide different performances and accuracy. The GN is designed to distribute the input data and select the optimal model for each node to meet the global accuracy and performance requirements of an inference request. As shown in Figure 3 (a), our framework saves an input requests queue in the GN as a tuple vector, including inference request number R, required performance P, and required accuracy A. The GN takes a request R1 from the queue, partitions it to four workloads (w_1,w_2,w_3,w_4) with an associated model number, and distributes it over the edge.

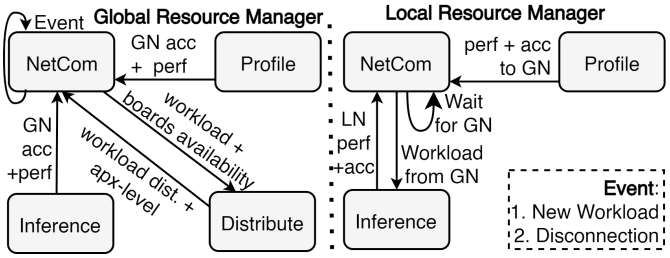

Figure 4: State Machine of GN and LN Resource Manager.

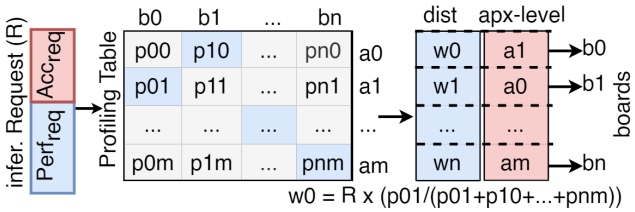

Figure 5: Workload distribution approach.

B. Resource Manager

We designed a distributed *Resource Manager* comprising a global module on the GN and a local module on each LN. The behavior of both modules is represented as a Finite State Machine (FSM) in Figure 4.

Global Resource Manager. The *Global Resource Manager* deployed in the GN has four states, viz. *Profile, NetCom, Distribute* and *Inference*. In the *Profile* state, we record the performance and accuracy of the GN at different approximation levels with test data from the *ImageNet* dataset to populate a profiling look-up table. After profiling, the FSM transitions to *NetCom* state to gather the connection status and profiling data of the connected LNs using the *Network Module*. The FSM updates the profiling table with the received LN data and waits for an event of either workload arrival or board disconnection. In case of a new workload, the FSM receives the workload performance and accuracy requirements and transitions to the *Distribute* state. The *Distribute State* invokes the *Dispatch Policy* (Section III-C) to receive the workload distribution with specified approximation levels and transitions to the *NetCom* state. The *NetCom* state broadcasts the workload distribution with the associated approximation levels to the relevant LNs and transitions to the *Inference* state. In the *Inference* state, the FSM triggers local inference of the received workload partition with associated approximation level and transitions to the *NetCom* state. Finally, if a board disconnects at runtime, the FSM again transitions to the *Ditribute* state for new workload distribution and broadcasts the new distribution with updated approximation levels to all the available boards.

Local Resource Manager. The FSM starts with the *Profile* state to record the performance and accuracy with test data and moves to the *NetCom* state to send the profiled data to the GN. The LN waits till the workload and the required approximation level are received from the GN and moves to the *Inference* phase to execute the local inference. Finally, after the successful inference, the FSM moves back to the *NetCom* state to send the output accuracy and performance to the GN.

C. Dispatch Policy for Workload Distribution

The *Dispatch Policy* is a GN sub-process that takes insights from the profiling table recorded by the *Resource Manager* to provide the run-time workload distribution and approximation.

Problem Formulation. The *Global Resource Manager* records the performance values of all available boards $\mathtt{b}_0, \mathtt{b}_1, \ldots \mathtt{b}_n$ in a profiling table $\mathtt{profiling\_table}_{\mathtt{mxn}}$ at different accuracy levels $a_0, a_1, \ldots, a_m$, where $\mathtt{n}$ is the number of boards and $\mathtt{m}$ is the lowest accuracy/highest approximation level. The columns of the profiling table represent the connected boards and the ascending row number shows the increasing approximation level as shown in Figure 5. The *Dispatch Policy* is invoked when a new inference request R with requirements of performance $\mathtt{Perf}_{\mathtt{req}}$ and accuracy $\mathtt{Acc}_{\mathtt{req}}$ arrives in the system. The sum of recorded performances at each row represents the combined edge cluster performance at a given approximation level. The *Dispatch Policy* calculates the percentage performance of each board against the cluster performance and proportionally splits the total workload performance requirement $\mathrm{perf}_{\mathrm{req}}$ as $\mathtt{perf}_{\mathtt{req}} = \sum_{i=1}^{n} \mathtt{perf\_b}_{\mathtt{req}}$ for each board. The *Dispatch Policy* is expected to find the required performance and accuracy by selecting the performance values in the profiling table closest to the per-board performance requirements without aggressively approximating the workload and return the workload distribution as w_dist = $\mathrm{w}_0, \mathrm{w}_1, \ldots, \mathrm{w}_n$ with the associated approximation levels.

Heuristics Model. The *Dispatch Policy* prunes the exploration space to meet the workload requirements at a minimum accuracy loss. Algorithm 1 shows that the policy copies the data of the `profiling_table` in a `pruned_table` while ignoring the disconnected boards (Lines 3–5). The policy saves the cluster performance at each approximation level in a `perf_vector` and stops when the recorded performance is equal to or higher than the required performance (Lines 6–9). It deletes all the remaining performance records at higher approximations to further reduce the exploration space (Lines 10–11). The policy calculates the board-level performance requirements and saves them in a vector $\mathtt{perf\_b}_{\mathtt{req}}\mathtt{[n]}$ (Lines 12–13). Then, it sends $\mathtt{perf\_b}_{\mathtt{req}}\mathtt{[]}$ and `pruned_table[][]` to a dynamic programming algorithm DP_{alg} to find the recorded performance values that are closest to the board-level performance requirements (Line 14). In this context, we used a standard subset sum algorithm for an efficient recursive search with time complexity `O(n * m)`. The algorithm starts with the values of the highest approximation level and back-propagates row-by-row to reach the closest values. The algorithm returns the performance distribution vector `P_dist` and the approximation vector `apx_dist` for each board. Finally, the policy calculates workload distribution vector `W_dist[]` by splitting the input data of the inference request R proportional to the performance factor of each board (Lines 15–16).

IV. Results and Evaluation

A. Experimental setup

We deployed our framework on a cluster of four devices including, Odroid XU4 (2x), Jetson Nano, and Raspberry Pi4.

Algorithm 1 Dispatch Policy

Input: profiling_table, avail_boards, R, Perf_{req}, Acc_{req}
procedure HEURISTICS MODEL
  for (m = 1 to max_apx_level) **do**
    for (n ∈ avail_boards) **do**
      pruned_table[m][n] = profiling_table[m][n]
  for (index = 1 to m) **do**
    total_perf_vector[index] = $\sum_{k=1}^{n}$ pruned_table[index][k]
    if perf_vector[index] ≥ perf_{req} **then**
      break;
  for (m > index) **do**
    pruned_table[m][].delete()
  for ($i \leftarrow 0$ to n) **do**
    $\text{perf\_}b_{\text{req}}[i]$ = perf_{req}*pruned_matrix[0][i]/perf_vector[0]
  P_dist[], apx_dist[] = DP_{alg}($\text{perf\_b}_{\text{req}}$[],pruned_table[m][n])
  for l = 1 to n **do**
    w_dist[l] ← R * $\frac{\text{P\_dist[l]}}{\sum_{k=0}^{n} P\_dis[]}$

Power Handling. Following the technical datasheet of each board, we have set the TDP limits of Odroid XU4, Raspberry Pi4, and Jetson Nano as 8W, 9W, and 10W. The Jetson Nano is equipped with onboard power sensors that we read at run-time for power monitoring. For Odroid XU4, we used *Smart Power-3* analyzer [14], and for Raspberry Pi4, we used the external shunt resistor method as used in [15] to monitor power.

Middleware Prototype. Figure 3 shows the used setup where we have implemented the framework as a middleware in C++, and each board has a running Linux Ubuntu 20.04. We have implemented a POSIX-based client-server architecture connected via Ethernet, including multi-threaded server operations and static IP-based identification to enable data and command-sharing between the edge nodes. We implemented a generic macro-based source code for all boards with 10 source files, 7 header files, and about 2000 lines of source code.

Workload Application. We have considered image classification as the baseline DNN application for our experimentation. The workload consists of a batch of images with labeled performance and accuracy requirements. The GN partitions the number of images following the *Dispatch Policy* (section III-C) and distributes them among the available nodes. We use popular Tensorflow-Lite [7] and OpenCV-Lite [16] libraries to load the input image and perform inference through pre-trained MobileNetV2 machine-learning models.

Accuracy Configuration We enforce accuracy configuration of DNN models through a pre-trained model selection method similar to [5] and [17]. We used the six different MobileNetV2 models that are trained on the ImageNet dataset and are available online at the official Tensorflow-lite repository [7]. These models are highly tunable and can achieve a range of accuracy and complexity trade-offs by adjusting the multiplier width parameter (alpha = 0.35, 0.5, 0.75, 1.0, 1.3, 1.4). Figure 6 presents the workflow of accuracy configuration through model selection. The *Resource Manager* selects the model at run-time as determined by the *Dispatch Policy*, and records the resultant accuracy and performance of the selected model.

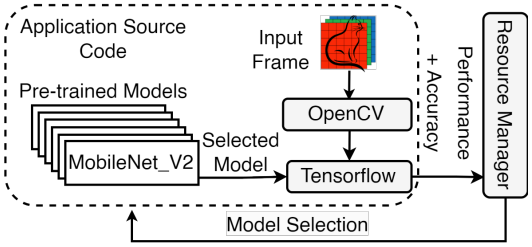

Figure 6: Run-time approximation in the Application Layer

B. Experimental results

Evaluation metrics We evaluate (i) performance as the number of inferences made per second and (ii) top-5% output accuracy by comparing predicted labels to true labels and calculating the percentage of correctly classified samples.

Comparison baseline We compared our results against the state-of-the-art workload distribution strategies including [9] as *Uniform* strategy for its even workload distribution in homogeneous edge clusters, [3] as *asymetric* approach for its non-uniform-workload distribution in heterogeneous edge cluster, and [5] as *Uniform + Apx* distribution approach for doing uniform workload distribution while considering workload approximation. We have profiled the performance of each device in terms of inference per second to formulate the profiling table. We have enhanced all of the compared strategies with DVFS actuation to avoid the TDP violation of each respective hardware board. We have only considered the workload distribution technique of these strategies and applied it to the input data partitioning for our implementation. We have also improved each strategy to meet the varying conditions of the workloads continuously.

Varying Workload Scenarios. We compared each strategy under dynamic workload variations i.e. different workloads with variable performance and accuracy requirements, and input batch sizes. We present the output performance and accuracy of each experiment in Figure 7 (a) and (b), respectively. We experiment with four different input batch sizes, and three different performance and accuracy requirements for each batch. The results show that the *Uniform + Apx* strategy meets the performance for each experiment while violating the minimum accuracy target due to aggressive approximation of the given workload. The *Asymmetric* and *Uniform* strategies generate more accurate results but are unable to meet the performance thresholds because the requirements are beyond the rated capabilities of the given edge cluster. Finally, the proposed strategy finds the optimal partitioning and approximation ensuring that the performance thresholds are met at the maximum output accuracy. Our approach yields an average performance gain of 52.63% and an accuracy improvement of 3.9% for the given set of experiments. Figure 8 shows the average performance violations (%) of *Uniform* and *Asymmetric* strategies and accuracy violations (%) of *Uniform + Apx* strategy under the aforementioned workload variation. We calculated the violation as the amount of time a strategy is unable to meet the target performance or accuracy in an execution cycle. The proposed strategy minimizes performance

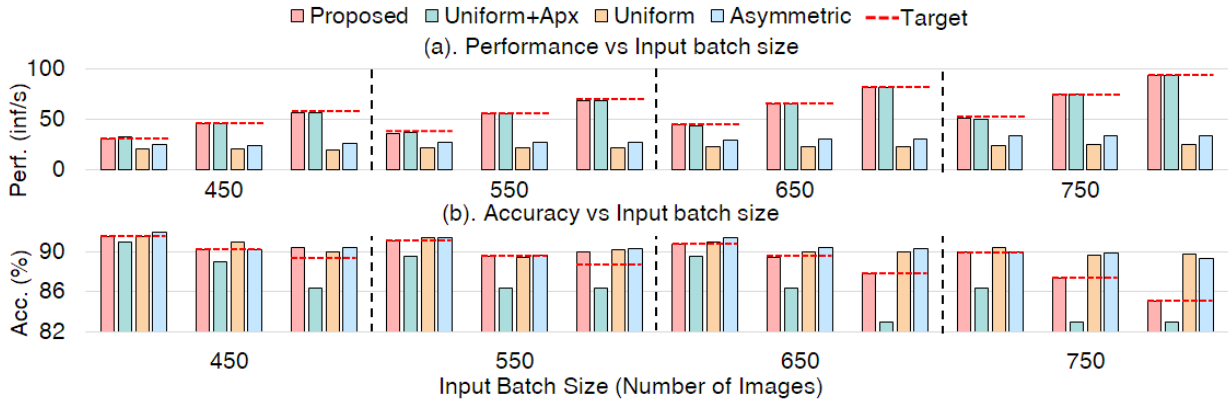

Figure 7: (a). Performance and (b). Accuracy (%) with dynamic workload variation. Each combination has different performance and accuracy requirements, and input batch sizes.

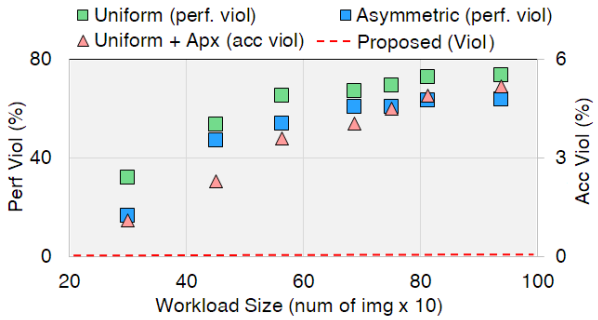

Figure 8: Performance and accuracy violations with varying input size (acc. viol. concern only uniform+apx and proposed).

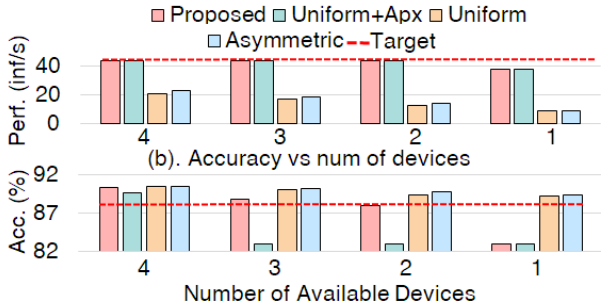

Figure 9: (a) Performance (inference per second) vs. available devices and (b) Accuracy (%) vs. available devices.

and accuracy violations on average by 41.52% and 5.2%.

Varying Device Availability. Figure 9 shows how each strategy copes with device unavailability/disconnection while meeting specific workload requirements. We specified a batch size of 650 images and established workload requirements that exceed the capacity of a single device in our setup. We progressively disconnect one device from the cluster at run-time. The proposed approach meets both performance and accuracy requirements by opportunistically exploiting accuracy tradeoffs. The *Uniform* and *Asymmetric* strategies fail to meet the performance requirements. The *Uniform+apx* strategy meets the performance requirements with two devices; however, it fails to meet accuracy constraints with greedy approximation. On average, our approach is showing output performance and accuracy improvements by 63.98% and 4.2%.

V. Conclusion

We propose an adaptive workload distribution policy that partitions DNN inference requests on collaborative heterogeneous edge clusters. Our approach exploits accuracy-performance trade-offs of DNN models to jointly determine optimal partitioning points and accuracy levels of dynamic inference requests. We implemented our strategy on a real hardware testbed of a cluster of heterogeneous edge devices. We evaluated our proposed approach against other relevant workload distribution strategies and observed an average performance gain of 42.5% and an average accuracy gain of 4.2% against accuracy-aware solutions.

VI. Acknowledgements

This work was supported by Marie Skłodowska Curie grant No. 956090 (APROPOS), Nokia Foundation and Kaute Saatio.

References

[1] X. Guo *et al.*, "Automated exploration and implementation of distributed cnn inference at the edge," *IEEE IoT Journal*, vol. 10, no. 7, 2023.
[2] A. Goel *et al.*, "Efficient computer vision on edge devices with pipeline-parallel hierarchical neural networks," in *2022 ASP-DAC*, 2022.
[3] K. Choi *et al.*, "Legion: Tailoring grouped neural execution considering heterogeneity on multiple edge devices," *IEEE ICCD*, 2021.
[4] Y. G. Kim *et al.*, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," *MICRO*, 2020.
[5] S. Shahhosseini *et al.*, "Online learning for orchestration of inference in multi-user end-edge-cloud networks," *ACM TECS*, dec 2022.
[6] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.
[7] TensorFlow, "Tensorflow lite," https://www.tensorflow.org/lite, 2017.
[8] R. Hall *et al.*, "Secure multiple linear regression based on homomorphic encryption," *Journal of Official Statistics*, vol. 27, no. 4, p. 669, 2011.
[9] F. Xue *et al.*, "EdgeLD: Locally Distributed Deep Learning Inference on Edge Device Clusters," *IEEE CHPCC*, pp. 613–619, 2020.
[10] J. Mao *et al.*, "MoDNN: Local distributed mobile computing system for Deep Neural Network," *DATE*, pp. 1396–1401, 2017.
[11] Z. Zhuoran *et al.*, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE TCAD*, 2018.
[12] L. Zhou *et al.*, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *IEEE SEC*, 2019, p. 195–208.
[13] Y. Huang *et al.*, "Enabling DNN Acceleration with Data and Model Parallelization over Ubiquitous End Devices," *IEEE IoT*, 2021.
[14] Harkernel, "Smart power 3," www.hardkernel.com/shop/smartpower-iii/.
[15] M. Feli, "An energy-efficient semi-supervised approach for on-device photoplethysmogram signal quality assessment," *Smart Health*, 2023.
[16] J. Doe and J. Smith, "Opencv lite: An efficient alternative for computer vision on resource-limited embedded systems," *IEEE ESL*, 2023.
[17] N. Moothedath *et al.*, "Online algorithms for hierarchical inference in deep learning applications at the edge," *arXiv*, pp. arXiv–2304, 2023.