HiDP:Hierarchical DNN Partitioning for Distributed Inference on Heterogeneous Edge Platforms

Zain Taufique
University of Turku
Turku, Finland
zatauf@utu.fi

Aman Vyas
University of Turku
Turku, Finland
amvyas@utu.fi

Antonio Miele
Politecnico di Milano
Milan, Italy
antonio.miele@polimi.it

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

Anil Kanduri
University of Turku
Turku, Finland
spakan@utu.fi

***Abstract*—Edge inference techniques partition and distribute Deep Neural Network (DNN) inference tasks among multiple edge nodes for low latency inference, without considering the core-level heterogeneity of edge nodes. Further, default DNN inference frameworks also do not fully utilize the resources of heterogeneous edge nodes, resulting in higher inference latency. In this work, we propose a hierarchical DNN partitioning strategy (*HiDP*) for distributed inference on heterogeneous edge nodes. Our strategy hierarchically partitions DNN workloads at both global and local levels by considering the core-level heterogeneity of edge nodes. We evaluated our proposed *HiDP* strategy against relevant distributed inference techniques over widely used DNN models on commercial edge devices. On average our strategy achieved 38% lower latency, 46% lower energy, and 56% higher throughput in comparison with other relevant approaches.**

***Index Terms*—Edge AI, DNN inference, Heterogeneous systems**

I. Introduction

Deep Neural Networks (DNNs) enable a wide range of applications such as augmented reality, smart glasses, live video analytics, etc [1]. Such applications demand real-time low latency DNN inference over continuous streaming input data [2]. Offloading DNN inference to remote cloud servers can lead to unpredictable latency with communication penalty, while local edge devices have limited compute capabilities to provide low latency inference [3]. Edge inference techniques distribute the inference workload among a cluster of collocated edge nodes to improve DNN inference latency [3]–[5].

Existing distributed edge inference strategies partition the inference workload into *blocks* by splitting the layers of a DNN model [6]–[12] and/or the input data of the DNN inference request [3], [4], [13]–[16]. Subsequently, these *blocks* are distributed among collocated edge nodes, based on the compute capacity of an edge node [3], [4], [17] and the computation-communication ratio of a *block* [2], [5], [7]. Typically, edge nodes are composed of heterogeneous processing units including CPU, GPU, and Neural Processing Units (NPUs), exhibiting a high degree of compute diversity within the single node and across the edge cluster. However, existing distributed inference techniques make global decisions on DNN partitioning and distribution, without considering the core-level heterogeneity of individual edge nodes.

After the creation and distribution of *blocks*, aforementioned techniques rely on deep learning frameworks to schedule the inference service on a local edge device. Deep learning frameworks do not fully utilize the resources of a heterogeneous multi-core edge node, leading to sub-optimal inference latency. For example, TensorFlow [18] schedules inference on GPU by default, unless explicitly specified by the application to use other CPUs/NPUs [19], [20]. Running inference on a single processing unit (e.g. GPU) misrepresents the compute capacity of the local heterogeneous edge node, resulting in sub-optimal workload partitioning and distribution decisions made on a global level. This problem is emphasized on edge platforms with CPUs performing better than GPUs [21] [10], and while running CPU-friendly layers of DNN models [22]. Advanced inference strategies that consider core-level heterogeneity are tailored for inference on a single edge node [10] [22] [1]. Adapting heterogeneity-aware DNN scheduling techniques for distributed inference requires intelligent orchestration to jointly optimize DNN partitioning and workload distribution, along with infrastructural support for inter-node data and decision control transfer. However, existing distributed inference strategies lack such intelligent orchestration.

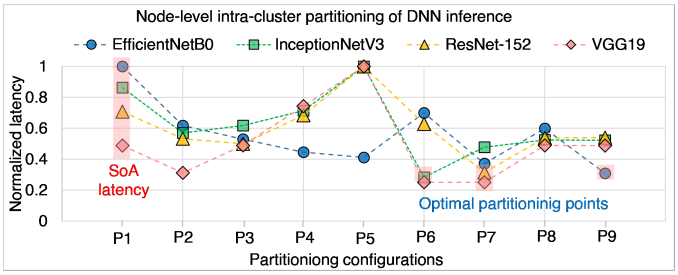

Fig. 1. Inference latency of DNN models with different workload partitioning configurations.

We demonstrate the limitations of existing distributed inference techniques over four DNN models run on the Jetson TX2 platform [23]. Figure 1 shows normalized inference latency of the DNN models with different workload partitioning configurations (`P1-P9`). Each partitioning configuration represents a specific combination of (i) number of data-wise partitions of the DNN model and (ii) CPU-GPU workload split. Among these configurations, P1 is the default TensorFlow workload scheduling choice – running inference exclusively on the GPU with no data partitioning. State-of-the-art distributed inference techniques use the workload partitioning configu-

ration P1 (highlighted as *SoA latency* in Figure 1) on a local edge device, resulting in higher inference latency. However, inference latencies of all the DNN models are lower in partitioning configurations other than P1, where the inference workload is partitioned and split between the CPU and GPU. Latency of ResNet-152 and VGG-19 are lowest at P7 (4 data partitions with 80% workload on GPU and 20% on CPU), InceptionNet-V3 at P6 (90% workload with 2 data partitions on GPU and 10% workload with 4 data partitions on CPU), and EfficientNet-B0 at P9 (4 data partitions and 50% workload split between CPU and GPU). InceptionNet-V3, ResNet-152, VGG-19, and EfficientNet-B0 have 65%, 40%, 25%, and 75% lower latency respectively, in comparison with existing distributed inference strategies using the default TensorFlow run-time. It should be noted that optimal partitioning configuration (number of partitions and workload split between CPU and GPU) differs for different DNN models. Our analysis highlights that the default partitioning configuration used by existing distributed inference techniques – (i) results in higher inference latency on local edge devices, and (ii) skews workload partitioning decisions exclusively made at a global level. These limitations can be addressed by considering the core-level heterogeneity of edge nodes while making both global and local workload partitioning decisions.

In this work, we propose a two-tier hierarchical DNN partitioning (*HiDP*) strategy to improve the latency of distributed inference over a cluster of heterogeneous edge devices. Our approach considers both core-level and device-level heterogeneity among edge devices within the cluster to combinatorially determine global DNN partitioning and workload assignment to edge nodes, followed by local DNN partitioning. To the best of our knowledge, ours is the first work that considers hybrid partitioning for distributed DNN inference in heterogeneous edge nodes while optimizing core-level resource usage. Our novel contributions include:

- Hierarchical DNN partitioning strategy for minimizing latency of distributed inference on edge platforms
- Collaborative edge cluster framework for partitioning, distribution, and scheduling of DNN inference services
- Evaluation of HiDP strategy against existing distributed inference techniques on real hardware edge cluster setup including Jetson Orin NX, Jetson Nano, Jetson TX2, Raspberry Pi 4B, and Raspberry Pi 5.

The paper is organized as follows: Section II provides background and motivation for our proposed approach, and Section III presents an overview of our framework infrastructure. Section IV evaluates our proposed solution against other relevant strategies, followed by conclusions in Section V.

II. Background and Motivation

A. DNN partitioning

For distributed inference, DNNs can be partitioned model-wise [4] and data-wise [16]. In *model partitioning*, DNN layers are dynamically grouped into executable *blocks*; these blocks are offloaded to multiple devices for distributed execution in a pipelined fashion. This strategy is inherently temporal since layers across different DNN *blocks* are executed sequentially. Minimizing inference latency requires creation of variable-sized DNN *blocks* dynamically by considering resources across the edge cluster. Model partitioning is feasible for dense DNN models that enable coarse creation of compute-intensive *blocks*. Alternatively, *data partitioning* splits the input data and creates smaller-sized sub-models of the original model for parallel execution. Data partitioned inference is spatio-temporal, since parallelly executed sub-models exchange intermediate data to maintain accuracy. Workloads with larger input sizes are suitable for data partitioning, considering the computation-communication ratio of intermediate data sharing. Minimizing inference latency of a DNN model on edge clusters requires joint selection of feasible partitioning strategy, optimal partitioning points, and workload distribution and scheduling.

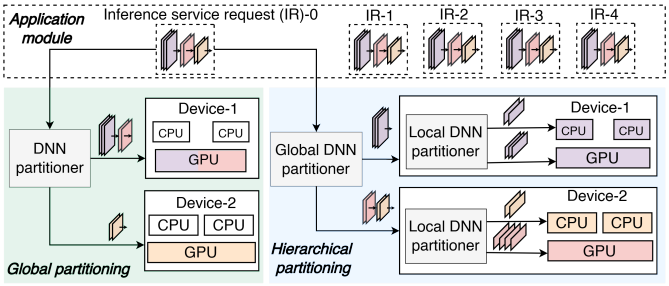

Fig. 2. Comparison of global and hierarchical DNN partitioning strategies.

B. Motivational Example

We demonstrate the benefits of hierarchical DNN partitioning in comparison with global partitioning strategy over a motivational scenario. Figure 2 shows an exemplar distributed DNN inference scenario, with a sequence of DNN inference requests (IR0-IR4) to be executed on two heterogeneous edge nodes, device-1 and device-2. With the global partitioning strategy (shown in green in Figure 2), the *DNN Partitioner* creates layer-wise blocks and distributes them among devices 1 and 2. Within devices 1 and 2, the inference blocks are executed on GPU by default. As presented empirically in Figure 1, this partitioning configuration leads to longer inference latency, potentially affecting other DNN inference requests in the queue. With the hierarchical partitioning strategy (shown in blue in Figure 2), the *Global DNN Partitioner* creates layer-wise blocks based on the core-level heterogeneity of devices 1 and 2. It should be noted that the hierarchical partitioning strategy creates blocks that are different from the global partitioning strategy. Subsequently, the *Local DNN Partitioner* on each device further partitions the DNN blocks and allocates to all the available cores. Thus, the hierarchical partitioning strategy results in lower inference latency, while also freeing up resources for subsequent inference requests in the queue. This example scenario highlights the need for making heterogeneity-aware DNN partitioning decisions globally, followed by local optimization for minimizing the inference latency.

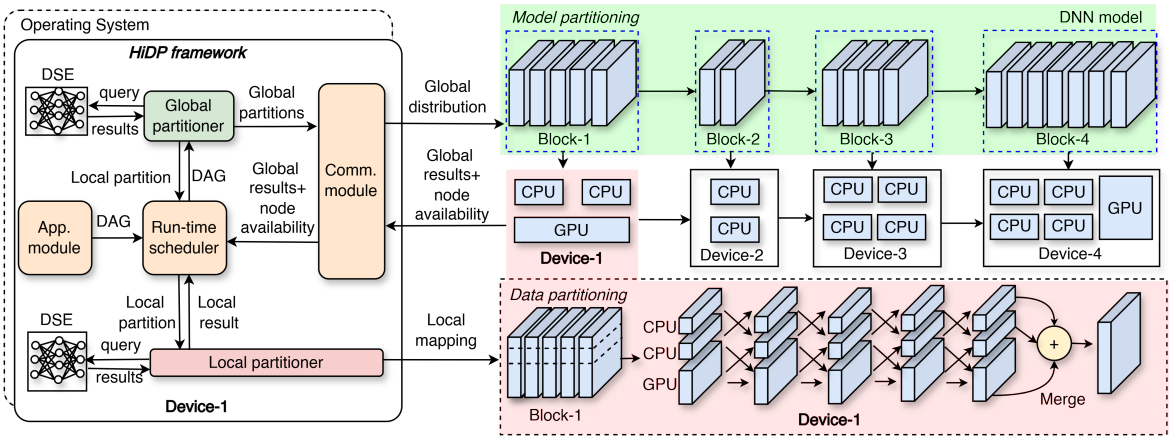

Fig. 3. Overview of the proposed *HiDP* framework. In this instance, the framework is run on device-1, partitioning the DNN model globally through model partitioning and locally through data partitioning.

C. Related Work

Existing distributed inference techniques use DNN model partitioning to schedule inference workload over multiple edge nodes in a pipelined fashion [1], [10]–[12] or offload the inference service to resourceful cloud [6]–[9]. Pipelining the model partitioned DNN inference is sequential and is beneficial for dense DNN models with a continuous stream of inference requests. Other techniques focus on data partitioning by splitting the input data into batches [17], sub-images [4], [15], [16], or intermediate layers through Fused Tile Partitioning (FTP) [3], [14]. Data partitioning allows parallel and distributed inference. However, the communication overhead of intermediate data exchange becomes significantly high for DNNs with a smaller input size. Recently, DisNet [5] proposed hybrid partitioning for DNN inference to minimize the overheads of both partitioning techniques without considering granular control over local device resources. *HiDP* performs hybrid partitioning using model and data partitioning while having fine-grained control over local edge node's resources to minimize inference latency. Table I compares *HiDP* against different workload partitioning strategies for edge clusters. Unlike the State-of-the-Art (SoA) strategies, *HiDP* considers latency and energy reduction through hierarchical partitioning.

III. HiDP Framework

HiDP is a 2-step workload splitting framework for distributed DNN inference across heterogeneous edge clusters. The nodes are connected via wireless networks and can collaborate at run-time to perform inference tasks. The DNN inference requests arrive randomly at a local node and the *HiDP* framework performs hierarchical partitioning to achieve low inference latency. As shown in the left side of Figure 3, the framework includes *Application Module*, *Communication Module*, *Global partitioner*, *Local partitioner*, and a *Run-time scheduler*. The right side of the figure shows the scenario of distributed DNN inference using *HiDP* framework across a heterogeneous cluster of four edge nodes. The *HiDP* framework receives a DNN inference request in the *Application module* of *Device-1* and sends the DNN to *Run-time scheduler*.

TABLE I
COMPARISON OF HiDP WITH OTHER RELEVANT APPROACHES

	Partition type	Target platform	Global Partitioning	Local Partitioning	heterogeneous block size
[3]	Data	Edge cluster	✓	✘	✘
[15]	Data	Edge cluster	✓	✘	✓
[7]	Model	Edge-cluster	✓	✘	✓
[9]	Model	Edge-cloud	✓	✘	✓
[5]	Hybrid	Edge cluster	✓	✘	✓
HiDP	Hybrid	Edge cluster	✓	✓	✓

We designed a scheduling policy in the *Run-time scheduler* module that monitors and controls the workload splitting and distribution across the edge cluster. The *Run-time scheduler* gets the status of the cluster-wide node availability and invokes the *Global partitioner* to find the optimal partitioning point. The *Global partitioner* consults a Design Space Exploration (DSE) agent to find the optimal partitioning mode and the feasible partitions. The *Global partitioner* selects model partitioning; then it distributes the workload across the edge cluster via *Communication Module*. The *Communication Module* provides access to send and receive data across the edge cluster. The *Global partitioner* sends the local partition to the *Run-time scheduler* for local execution. The *Run-time scheduler* invokes the *Local partitioner* to find the optimal partitioning point and mode using a *DSE* agent. The *Local partitioner* selects data partitioning and splits the workload following heterogeneity of two CPUs and one GPU. The *Run-time scheduler* gets the local and global results via *Communication Module* and merges all the results.

Platform. The edge nodes are supported by an Operating System (OS) with relevant libraries and programming framework to run DNN inference, allowing inter-node communication, workload scheduling, and application-to-core mapping. The OS provides interfaces between software-software modules to exchange data, and software-hardware modules to bind applications to selected processors. Unlike SoA strategies, *HiDP* overtakes the control from default OS governors and allocates the workload to the desired processing units. The OS allows run-time monitoring of the board's power consumption through onboard sensors or external power monitoring equipment to measure the energy consumption of a DNN inference.

Workloads. We design *HiDP* to handle DNN inference requests without prior knowledge of the workload arrival time.

We target applications with streaming inputs of different sizes and batches. We consider the modern example scenario of a person bearing different smart gadgets and wearables including a smartwatch, smartphone, smart ring, and augmented reality gear. Manufacturers like Apple and Samsung have produced a series of smart devices that can communicate with each other at run-time sharing notifications and live data for a single user. These devices have diverse DNN applications that perform cognitive vision tasks of variable input sizes and data volume using similar DNN models depending on the requirements of the application tasks.

System Model. We consider DNN model as Directed Acyclic Graph (DAG) since the data flow is sequential without loops and each partition is executed only once. The DAG nodes represent the DNN layers and the edges represent the tensors. The DNN is denoted as $\mathcal{D}(\mathcal{L}_i) = \{\mathcal{L}1, \mathcal{L}2, ..., \mathcal{L}_i\}$, where $\mathcal{L}$ represents set of layers that can be partitioned following the model and data partitioning. The layer types include convolution, pooling, flatten, or dense, where each layer can be represented as a vector of kernel size, stride, padding, number of input channels, number of output channels, and height of the input dimensions. We denote edge cluster as $\mathcal{N}(\phi_j) = \{\phi 1, \phi 2, ..., \phi_j\}$, where ϕ_j represents the edge node. For each node, there exist k processors such that $\phi = \{\rho_1, \rho_2, ..., \rho_k\}$ where ρ_k represents CPU, GPU, or NPU. We denote the computation frequency of a processor as f_k as computation cycles per second. We define the compute intensity of a DNN as δ, representing the average number of compute cycles of a processor to execute 1 flop [cycles/flops]. We define the computation rate [flops/sec] of each processor as the ratio of the computation frequency of the processor to the compute intensity of the DNN, such as $\lambda = \frac{f_k}{\delta}$ [24]. We denote the communication rate of each processor as a scalar μ_k, representing the DNN transmission overhead between two processors for a given time duration t. We calculate the local computation-to-communication ratio of a node as:

$$\psi\{\lambda, \mu\} = \left\{ \ \frac{\lambda_1}{\mu_1}, \quad \frac{\lambda_2}{\mu_2}, \quad \cdots \quad , \frac{\lambda_k}{\mu_k} \ \right\} \tag{1}$$

Finally, we calculate the computation rate Λ_j of a node ϕ_j as the sum of computation rates of the available processors: $\Lambda_j(\rho_k) = \sum_1^k [\lambda_k]$. We denote the communication rate of each node as a scalar β_{ϕ_j} representing the DNN data transmission overhead between two nodes for a given time duration t via a wireless network. *HiDP* calculates the communication rate of each node by sending a set of pseudo packets to each node and recording the time taken to get the response. We form a global resource vector Ψ including the global computation-to-communication ratio of all nodes such that:

$$\Psi\{\Lambda, \beta\} = \left\{ \ \frac{\Lambda_1(\rho_k)}{\beta_{\phi_1}}, \quad \frac{\Lambda_2(\rho_k)}{\beta_{\phi_2}}, \quad \cdots \quad \frac{\Lambda_j(\rho_k)}{\beta_{\phi_j}} \ \right\} \tag{2}$$

HiDP formulates an availability vector $\mathcal{A}(\mathcal{N}_\phi)$ based on the communication rate $\beta_\phi^{(t)}$ such that:

$$\mathcal{A}(\mathcal{N}_\phi) = \{\alpha_1, \alpha_2, \cdots, \alpha_j\} \quad , \quad \alpha_j = \begin{cases} 1 & \text{available,} \\ 0 & \text{unavailable} \end{cases} \tag{3}$$

Algorithm 1 HiDP framework on leader node

1: $\mathcal{D}_\mathcal{L}$ arrives on ϕ_1 //Get the input DAG
2: $\phi_1 \leftarrow \phi^*$ //Assign leader status
3: $\phi_1 \leftarrow \mathcal{A}(\mathcal{N}_\phi)$ // Get availability status
4: $\Theta_\omega \leftarrow \text{DP}_{alg}(\omega, \Psi(\phi_j, \beta))$ //latency of model partitioning
5: $\Theta_\sigma \leftarrow \text{DP}_{alg}(\sigma, \Psi(\phi_j, \beta))$ //latency of data partitioning
6: $\Theta \leftarrow \min(\Theta_\omega, \Theta_\sigma)$ //decide partitioning mode
7: ϕ_1 partitions $\mathcal{D}_\mathcal{L}$ and allocates to nodes
8: $\theta_\omega \leftarrow \text{DP}_{alg}(\omega, \psi(\rho_k, \mu_k))$ //latency of model partitioning at local
9: $\theta_\sigma \leftarrow \text{DP}_{alg}(\sigma, \psi(\rho_k, \mu_k))$ //latency of data partitioning at local
10: $\theta \leftarrow \min(\theta_\omega, \theta_\sigma)$ //select partitioning mode at local
11: ϕ_1 executes local workload
12: $\phi_1 \leftarrow$ local and global results
13: ϕ_1 merges the results and reports prediction

For workload partitioning, *HiDP* decides between the partitioning modes and distributes the workload to the resources. For model partitioning, we denote the width of a layer block as ω_i and calculate the total computation time as:

$$\Theta_\omega = \left\{ \ \gamma \cdot \omega \ \right\} \quad , \quad \gamma = \begin{cases} \Psi & \text{for global partitioning} \\ \psi & \text{for local partitioning} \end{cases} \tag{4}$$

Similarly, *HiDP* explores the number of parallel submodels σ for data partitioning to calculate the total computation time as:

$$\Theta_\sigma = \left\{ \ \gamma \cdot \sigma \ \right\} \quad , \quad \gamma = \begin{cases} \Psi & \text{for global partitioning} \\ \psi & \text{for local partitioning} \end{cases} \tag{5}$$

HiDP calculates the total computation time Θ of both model and data partitioning modes and selects the faster strategy.

Scheduling Algorithm. Algorithm 1 shows the high-level workflow of *HiDP* framework to perform distributed inference. *HiDP* assigns leader status (ϕ_1^*) to the node that receives new inference request $\mathcal{D}_\mathcal{L}$ (Lines 1–2). The leader node checks the availability status ($\mathcal{A}(\mathcal{N}_\phi)$) of the cluster (Line-3) and finds the optimal partitioning mode using Dynamic Programming (DP) algorithm (Lines 4–6). We used a standard subset sum algorithm for an efficient recursive search with time complexity $O(n * m)$, where n represents the number of DNN *blocks* and m represents the number of available nodes. The algorithm starts with the largest possible block sizes following the resource heterogeneity to calculate the inference latency and back-propagates block by block to converge to minimum latency. We used the same algorithm to explore global and local partitioning points because the function arguments are essentially the same in either case including the DNN and the computation-communication ratio. The leader node partitions the workload and distributes the partitions to the global nodes (Line 7). The leader node finds out the optimal partitioning mode and the partitioning point for the local partition (Lines 8–10). The leader node executes the local workload on its local processors and gathers the global results (Lines 11–12). The leader node merges the final results and reports the prediction to the DNN application (Line 13).

Run-time Scheduler. We have designed the scheduling policy of the *Run-time Scheduler* as a Finite State Machine (FSM) including *Analyze, Explore, Offload, Map,* and *Execute* states

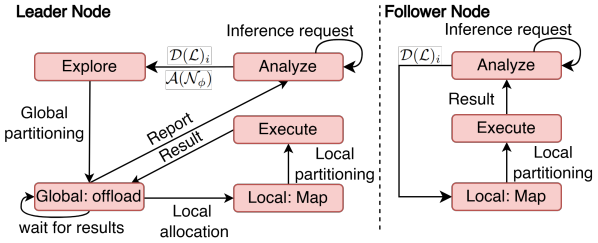

Fig. 4. Workflow of the leader and follower nodes in the *HiDP* controller

as shown in Figure 4. The scheduling policy is the implementation on the method explained in Algorithm 1. The scheduling policy is different for the leader and follower nodes.

Leader Node. In the *Analyze* state, the controller waits for an inference request from a DNN application in the *Application module*. When a request is triggered, the controller checks the availability status of the cluster nodes by sending and receiving a status packet to the nodes via *Communication module* and transitions to the *Explore* state. In the *Explore* state, the controller refers to the global DSE agent to find the optimal partitioning point of the given DNN workload for global distribution. After finding the optimal partitioning point, the controller transitions to the *Global: offload* state. The controller offloads the workload to the available nodes using the *Communication module* and transitions to the *Local: Map* state for local execution of the allocated workload. Here, the controller refers to the local DSE agent to figure out the local partitioning of the workload following the available processing units. After converging to the optimal partitioning point, the controller transitions to the *Execute* state. In this state, the controller executes the workload while sharing intermediate data with the available nodes for parallel or sequential execution, depending on the partitioning mode. After successful execution, the controller gathers the results and transitions back to the *Global: offload* state for final merging and reporting of the results. After merging the results from local execution and cluster nodes, the controller transitions to the *Analyze* state and waits for the next inference request.

Follower Nodes. For the follower nodes, the state machine is much simpler, such that (i) the node receives the distributed workload from the leader node in the *Analyze* state, (ii) executes it after local partitioning in the *Local: Map* and the *Execute* state, and (iii) report back the results to the leader.

IV. Experimental Evaluation

A. Experimental Setup

Evaluation Platform. We design a collaborative heterogeneous edge cluster, comprising commercial edge platforms (shown in Table II), and deploy the *HiDP* strategy on the defined platform. We monitor the run-time power consumption of the Jetson boards using the on-board power sensors. We use the external shunt resistor to monitor the power consumption of the Raspberry Pi boards.

Workloads. We evaluated our framework over widely used DNN models of ResNet152, EfficientNetB0, VGG-19, and InceptionNetV3. We implemented these models using the TensorFlow library [18], with input image sizes of 224x224, and 299x299. We enhanced the top layer of these models to accept variable input sizes to enable data partitioning.

TABLE II
Technical specifications of the evaluation setup

Device	CPU	GPU	DRAM
Jetson Orin NX	8x ARM Cortex-A78	1024-core Ampere	8 GB
Jetson TX2	2x Denver-2, 4x ARM Cortex-A57	256-core Pascal	8 GB
Jetson Nano	4x ARM Cortex-A57	128-core Maxwell	4 GB
Raspberry Pi 5	2x ARM Cortex-A76	VideoCore VII	4 GB
Raspberry Pi 4	2x ARM Cortex-A72	VideoCore VI	4 GB

Middleware. We implemented *HiDP* as a middleware in Python with a source code of 400 lines. Each device hosts Linux 18.04 OS to provide software and hardware interfaces. We used CGroup libraries to bind the workload to the desired number of CPU cores. For GPU implementation, the TensorFlow backend used CUDA libraries for NVIDIA boards and OpenCL for Raspberry Pi boards. All the devices are connected over 80 MBps wireless control through POSIX-based client-server architecture. We used multi-threaded server operations for the leader nodes to communicate with the available nodes dynamically. The overhead of using DP algorithm-based exploration including both global and local partitioning is 15ms on average for our experimentation.

Comparison w.r.t. state-of-the-art approaches. For evaluation, we considered three different state-of-the-art partitioning strategies – data [4], model [7], and hybrid [5]. *MoDNN* [4] partitions and distributes the input data proportionally among the available edge nodes. We implemented *MoDNN* using the data partitioning module of *HiDP* framework. OmniBoost [7] determines the optimal partitioning point using the Monte-Carlo search tree and pipelines the DNN inference over both CPU and GPU. We implemented the throughput estimator of *Omniboost* using Gymnasium library [25] and trained it on our target workloads. *DisNet* [5] uses heuristic-based DNN partitioning and distribution by jointly considering data and model partitioning. We used the data and model partitioning algorithm of *HiDP* to implement *DisNet*.

B. Experimental Results

For our experiments, we consider inference latency, throughput, energy consumption (based on run-time power monitoring), and accuracy as evaluation metrics. Figure 5 (a) shows the inference latency of EfficientNetB0, InceptionNetV3, ResNet152, and VGG-19 using different strategies. Our proposed *HiDP* strategy has the lowest inference latency for all the workloads, in comparison with other relevant strategies, achieving upto 61%, 61%, 59%, and 49% lower latency for EfficientNet, InceptionNet, ResNet, and VGG, respectively. The hierarchical partitioning strategy of *HiDP* jointly optimizes both global-level DNN block creation and workload distribution, followed by local-level fine-grained partitioning and workload scheduling. This results in significantly lower latency compared to other distributed inference strategies that are exclusively confined to global partitioning. On average, *HiDP* has 37%, 44%, and 56% lower latency than *DisNet, OmniBoost,* and *MoDNN* strategies, respectively. Figure 5 (b) shows the energy consumption of different partitioning strategies for all the workloads. The lowest inference

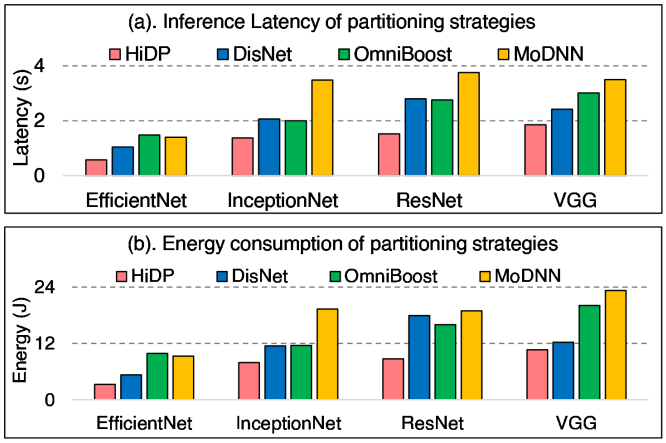

Fig. 5. Inference (a). latency and (b). energy consumption of different strategies for targeted DNN workloads.

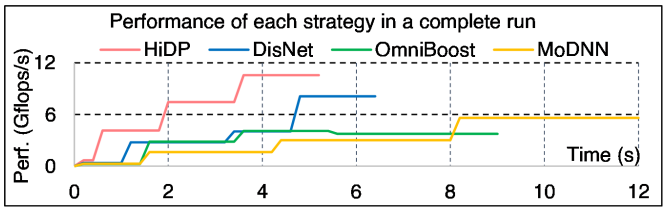

Fig. 6. Performance (Gigaflops/s) of each strategy while concurrently running targeted DNN workloads.

latency of *HiDP* strategy also reflects in the lowest energy consumption for all the workloads. *HiDP* consumes upto 67%, 59%, 54%, and 54% lower energy for EfficientNet, inceptionNet, ResNet, and VGG against relevant strategies. On average *HiDP* consumes 33%, 48%, and 58% lower energy than DisNet, OmniBoost, and MoDNN, respectively.

We evaluate the adaptability of different strategies under varying workload dynamics. We created a dynamic workload with successive run-time inference requests for every 0.5s, in the order of EfficientNetB0, InceptionNetV3, ResNet152, and VGG-19. This creates a progressively increasing workload such that at t=1.5s, all four DNNs are running concurrently on the edge cluster. Figure 6 shows the performance (Gigaflops per second) of each strategy while running different DNN models. It can be noticed that *HiDP* delivers the highest performance throughout the execution cycle. Lower latency achieved by *HiDP* frees up the resources of edge nodes, which can be efficiently used to service subsequent inference requests. *HiDP* completes the inference of all the models within 5s in total, achieving 39%, 54%, and 56% higher performance than *DisNet, OmniBoost,* and *MoDNN*, respectively. Higher per-inference latency of other strategies keeps the worker edge nodes busy for a longer duration, affecting the overall performance and throughput. We also evaluate the throughput (number of inferences per 100s) of different strategies over 8 different mixes of DNN inference requests. We created *Mix 1-4* and *Mix 5-8* with two and three different DNN models from the target workloads, respectively. *HiDP* achieves significantly higher throughput (upto 150% in *Mix-2* and 56% on average) compared to other strategies across all the workload mixes. *HiDP* dynamically selects data/model partitioning based on DNN characteristics, flexibly achieving low latency inference

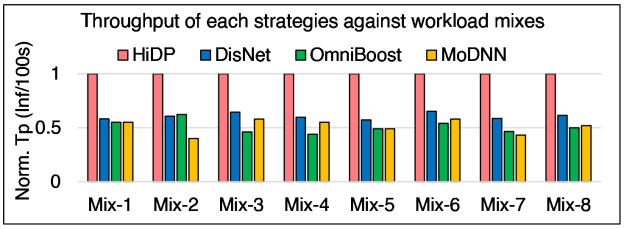

Fig. 7. Throughput of different strategies while running different combinations of targeted DNN workloads concurrently.

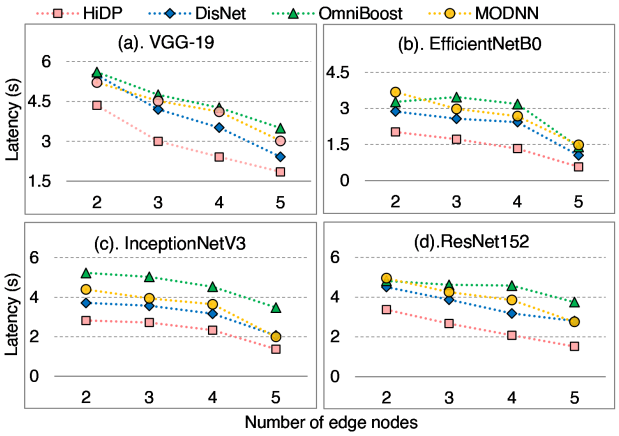

Fig. 8. Inference latency with varying number of worker edge nodes.

across different workload mixes. This is reflected in higher throughput achieved by *HiDP* in comparison with other strategies that are confined to specific partitioning configurations. We evaluate the adaptability of different strategies with varying numbers of edge devices within the cluster. Figure 8 shows inference latency of all the DNN workloads using different strategies with 2-5 edge nodes. For all the workloads and varying numbers of edge nodes, *HiDP* has the lowest inference latency. It should be noted that the latency gap between *HiDP* and other strategies becomes prominent with a decreasing number of worker edge nodes. This can be attributed to *HiDP*'s efficient utilization of local edge node resources, while the other global strategies are affected by a lower number of edge devices. On average *HiDP* achieves 30%, 46%, and 38% lower latency than DisNet, OmniBoost, and MoDNN, respectively. For VGG-19, EfficientNetB0, ResNet-152, and InceptionNet-V3, *HiDP* has an average Top-1% accuracy of 75.3%, 77.1%, 78.6%, and 80.9% and Top-5 % accuracy of 89.7%, 92.25%, 92.7%, and 92.5%, respectively. Both the Top-1%, and Top-5% accuracies of *HiDP* are the same as DisNet, OmniBoost, and MoDNN, demonstrating robust intermediate data sharing while enforcing DNN partitioning.

V. Conclusion

We present *HiDP* framework for low latency DNN inference on distributed edge platforms. Our approach hierarchically partitions the DNN inference workload at a global level, followed by optimized partitioning and scheduling on a local heterogeneous edge node. We evaluated *HiDP* on four Jetson platforms and two Raspberry Pi platforms achieving latency and energy improvements of 38%, and 46% against SoA strategies, respectively.

REFERENCES

[1] J. Seong *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.

[2] J. Kim *et al.*, "Energy-aware scenario-based mapping of deep learning applications onto heterogeneous processors under real-time constraints," *IEEE Transactions on Computers*, 2022.

[3] Z. Zhuoran *et al.*, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[4] J. Mao *et al.*, "MoDNN: Local distributed mobile computing system for Deep Neural Network," *Proc. of Design, Automation and Test in Europe, DATE*, pp. 1396–1401, 2017.

[5] E. Samikwa *et al.*, "Disnet: Distributed micro-split deep learning in heterogeneous dynamic iot," *IEEE Internet of Things Journal*, vol. 11, no. 4, pp. 6199–6216, 2024.

[6] F. Cunico *et al.*, "I-split: Deep network interpretability for split computing," in *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE, 2022, pp. 2575–2581.

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

[8] S. K. Ghosh *et al.*, "Partnner: Platform-agnostic adaptive edge-cloud dnn partitioning for minimizing end-to-end latency," *ACM Transactions on Embedded Computing Systems*, 2023.

[9] A. Kosmas *et al.*, "Road-runner: Collaborative dnn partitioning and offloading on heterogeneous edge systems," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023.

[10] S. Wang *et al.*, "High-throughput cnn inference on embedded arm big. little multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, 2019.

[11] Y. Wu *et al.*, "Moc: Multi-objective mobile cpu-gpu co-optimization for power-efficient dnn inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–10.

[12] H. Li *et al.*, "Hasp: Hierarchical asynchronous parallelism for multi-nn tasks," *IEEE Transactions on Computers*, 2023.

[13] S. Zhang *et al.*, "Deepslicing: Collaborative and adaptive cnn inference with low latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[14] K. Choi *et al.*, "Legion: Tailoring grouped neural execution considering heterogeneity on multiple edge devices," in *IEEE Int. Conf. on Computer Design (ICCD)*, 2021, pp. 383–390.

[15] X. Guo *et al.*, "Automated exploration and implementation of distributed cnn inference at the edge," *IEEE Internet of Things Journal*, vol. 10, no. 7, pp. 5843–5858, April 2023.

[16] Y. Huang *et al.*, "Enabling DNN Acceleration with Data and Model Parallelization over Ubiquitous End Devices," *IEEE Internet of Things Journal*, 2021.

[17] Z. Taufique *et al.*, "Adaptive workload distribution for accuracy-aware dnn inference on collaborative edge platforms," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024.

[18] A. A. Abadi *et al.*, "Tensorflow," https://www.tensorflow.org/, 2015.

[19] TensorFlow Developers, *TensorFlow Guide: GPU support and Manual Device Placement*, 2023. [Online]. Available: www.tensorflow.org/guide/gpu/#manual_device_placement

[20] Princeton Research Computing, *TensorFlow on Princeton Research Computing Clusters*, https://researchcomputing.princeton.edu/support/knowledge-base/tensorflow, 2023.

[21] Y. G. Kim *et al.*, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," *Proc. of Int. Symp. on Microarchitecture, MICRO*, pp. 1082–1096, 2020.

[22] E. Aghapour *et al.*, "Arm-co-up: Arm co operative u tilization of p rocessors," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[23] NVIDIA, "Jetson tx2 module," 2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2

[24] L. Zeng *et al.*, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[25] M. Towers *et al.*, "Gymnasium," 2023. [Online]. Available: https://zenodo.org/record/8127025