IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 May 2024, accepted 31 May 2024, date of publication 3 June 2024, date of current version 13 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3409057

RESEARCH ARTICLE

DNNSplit: Latency and Cost-Efficient Split Point Identification for Multi-Tier DNN Partitioning

PARIDHIKA KAYAL, (Graduate Student Member, IEEE), AND ALBERTO LEON-GARCIA, (Life Fellow, IEEE)
Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada
Corresponding author: Paridhika Kayal (paridhika.kayal@mail.utoronto.ca)

This work was supported by the Doctoral Completion Award.

ABSTRACT Due to the high computational demands inherent in Deep Neural Network (DNN) executions, multi-tier environments have emerged as preferred platforms for DNN inference tasks. Previous research on partitioning strategies for DNN models typically involved leveraging all layers of the DNN to identify optimal splits aimed at reducing latency or cost. However, due to their computational complexity, these approaches face scalability issues, particularly with models containing hundreds of layers. The novelty of our work lies in uniquely identifying specific split points within various DNN models that consistently lead to efficient latency or cost partitioning. Under the assumption that per unit computing cost decreases in higher tiers and that bandwidth is not free, we show that only these specific split points need to be considered to optimize latency or cost. Importantly, these split points are independent of different infrastructure configurations and bandwidth variations. The key contribution of our work is the significant reduction in the computational complexity of DNN partitioning, making our strategy applicable to models with a large number of layers. Introducing *DNNSplit*, an adaptive strategy, enables dynamic split decisions in varying conditions with the least complexity. Evaluated across nine DNN models varying in size and architecture, *DNNSplit* exhibits exceptional effectiveness in optimizing latency and cost. Even for a more substantial model containing 517 layers, it identifies only 5 points as potential split points, thereby reducing the partitioning complexity by more than 100x. This makes DNNSplit especially advantageous for managing larger models. *DNNSplit* also demonstrates significant improvements for multi-tier deployments compared to single-tier execution, including up to 15x latency speedup, 20x cost reduction, and 5x throughput enhancement.

INDEX TERMS Cost-efficient, multi-tier, near-edge.

I. INTRODUCTION

DNN applications play a significant role in various domains, including image recognition, natural language processing, and recommendation systems [1]. These applications often require significant computational resources and can generate substantial data transmission requirements. The traditional approach of deploying DNN applications in the Cloud can result in significant delays when transferring input data from Edge devices to the Cloud. On the other hand, Edge devices have constrained resources, making it challenging to process DNN applications entirely on the Edge. To address this dilemma, hybrid multi-tier computing environments [2], [3], [4] have been adopted. The multi-tier architecture [3] consists of Edge devices, Near-Edge (NE) instances, and Cloud instances, each with varying computational capabilities and associated costs. Multi-tier computing environments are characterized by their dynamic nature and varying network conditions.

In a three-tier computing environment, DNN applications can be deployed through four main approaches: 1) Edge deployment, 2) NE deployment, 3) Cloud deployment, where the entire DNN model resides in the Edge device, NE instance, and Cloud, respectively; and 4) Multi-tier deployment, where the DNN model undergoes partitioning into two or three segments, with the initial layers deployed

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Gupta.

in the lower tiers and the higher layers in the upper tiers [5]. The upper tiers await the completion of computation in the lower tiers to receive the necessary output data for processing. Consequently, the processing of partitions in the upper tiers is contingent on the completion of computations in the lower tiers.

A critical concern is the strategic selection of partition points for DNN models to minimize the latency and system cost while meeting QoS and SLA constraints. However, the tremendous number of layers makes it difficult to design a latency and cost-efficient placement by considering all layers of the DNN. Therefore, it is essential to identify potential split points that, when used for partitioning, may lead to a latency or cost-efficient partitioning strategy for large DNNs consisting of several hundreds of layers.

The partitioning of DNN strikes a delicate trade-off between computation and communication [6]. As computation shifts to the upper tier, the time and costs of processing decrease, however, this comes at the expense of transmitting input data to the upper tier for processing. Therefore, the partitioning strategy is efficient only when the upper-tier processing time/cost reduction compensates for the increase in transmission time/cost. For DNNs, the output of some intermediate layers is significantly smaller than that of raw input data [7]. We leverage this insight to identify potential split points for various DNNs, laying the foundation for a cost-efficient splitting strategy in a multi-tier environment.

The split points identified by our strategy remain consistent regardless of the available bandwidth between tiers. Through evaluation in Section IV, we confirm that these points are the sole ones offering advantages, with no other split point yielding lower latency or cost. This novel discovery eliminates the necessity of individually examining all layers to determine efficient split points. Moreover, we leverage these points to develop a solution for latency and cost-efficient partitioning in a three-tier environment. We break down the problem into two steps, reducing complexity to linear.

Firstly, we determine the latency and cost-efficient tier for the single-tier execution of the DNN model. Next, we utilize the single-tier solution to address the problem of finding one or two split points to minimize latency and cost.

We develop an algorithm that allocates DNN applications efficiently to the most suitable tier, considering factors like computational capacity, data transmission delays, and associated processing and transmission costs. The goal is to minimize latency, reduce inference costs, and maximize the number of completed inference tasks per second within set QoS and Service Level Agreement (SLA) constraints. Additionally, our algorithms account for network and cost dynamics in partitioning decisions, adapting to parameter variations between tiers.

The document is structured as follows: Subsec.I-A outlines the study's motivation and contributions. Sec.II reviews existing research on DNN partitioning. Sec.III introduces our system model and the optimization problem. Sec.IV presents the strategy to find the network-oblivious potential split points effective for latency and cost minimization for multi-tier environments. The *DNNSplit* algorithm is explained in Sec.V. Evaluation is discussed in Sec.VII, and the paper concludes in Sec. VIII.

A. MOTIVATION AND PROBLEM STATEMENT

Existing research has primarily focused on the two-tier partitioning approach to minimize either inference time, computational cost, or both. However, much of the prior research lacks detailed insights into the critical layers essential for the success of their splitting algorithms. Given the substantial number of layers in DNN applications, considering each layer in the context of three-tier partitioning decisions proves impractical. Hence, there is a need to identify specific layers for inclusion in adaptive partitioning decisions.

Our contribution entails a comprehensive analysis of layers across different models, identifying the splitting positions capable of reducing latency and cost. This novel contribution is significant as these split points remain unaffected by different infrastructure configurations and bandwidth variations among the tiers. We demonstrate that our detected split points are the exclusive candidates for latency and cost improvements; no other split point can yield lower latency or cost. This unique contribution offers valuable insights for researchers seeking to streamline their partitioning strategies. To our knowledge, this work is the first to identify potential split points for various DNNs that are independent of network conditions and act as the sole candidates for latency and cost reduction. Our novel strategy comprehensively considers three different optimizations for DNN placement in a three-tier environment and yields favorable results with minimal time complexity.

We define two placement problems:

1) **Latency Minimization:** This approach aims to minimize the time required to complete the inference task to meet the QoS requirements, taking SLA into account.
2) **Cost Minimization:** This approach seeks to minimize expenses to adhere to budget limitations defined by SLA while meeting the application QoS requirement.

B. CONTRIBUTIONS

In this paper, our major contributions are as follows:

- We introduce a novel strategy, *DNNSplit*, to identify potential layers in various DNN models that can serve as effective split points for minimizing overall latency and execution costs. These split points remain independent of different infrastructure configurations and network conditions. Through our analysis, we confirm that these identified points are the only ones providing benefits, with no alternative split point resulting in lower latency or cost. This significant contribution reduces the computational complexity of DNN partitioning, making our strategy applicable to models with a large number of layers.

- Our splitting strategy involves two key steps: First, identifying the optimal tier for running the DNN model to minimize latency and cost while maintaining specified QoS and SLA constraints. Second, using the single-tier solution, our algorithm partitions DNN models for execution across different tiers in linear time, aiming to reduce latency and cost, all while ensuring QoS and SLA compliance. We also show the throughput improvement that can be gained using the *DNNSplit* strategy.
- Our partitioning strategy, evaluated across nine DNN models, demonstrates significant performance gains. Even for larger models consisting of 517 layers, our strategy identifies only 5 potential split points, reducing the complexity by more than 100-fold. For these larger models, we achieve up to a 15x latency speedup, 20x cost reduction, and 5x throughput improvement compared to single-tier execution.

II. RELATED WORK

Significant research has been conducted on collaborative methodologies aimed at improving DNN inference. These methods aim to partition the DNN model into two or three segments, distributed across different tiers, to reduce overall inference time. These approaches can be broadly classified into two-tier [7], [8], [9], [10], [12] or three-tier [8], [11], [13] joint inference acceleration techniques.

A. TWO-TIER DNN PARTITIONING

Neurosurgeon [7] focuses on finding the optimal splitting point for chaining DNNs to decrease energy consumption and latency. However, it does not consider deployment costs and the DAG structure of DNNs. DADS [12] is a Dynamic Adaptive DNN Surgery technique to segment DNN models, aiming to expedite DNN inference and increase throughput. DADS does not consider cost, and its computational complexity makes it impractical for larger models and multi-tier partitioning. Reference [10] addresses this challenge by reducing the complexity to a linear level. This modification enables experiments on more extensive models like Inception V4. However, the authors only explore 2-tier partitioning, recognizing that the complexity would become quadratic for a three-tier partitioning. Reference [9] developed an adaptive two-tier division algorithm to reduce both latency and cost. Notably, the algorithm doesn't account for bandwidth costs and propagation delay and assumes zero cost for Edge devices, overlooking fixed costs and operational costs. Alternative partitioning approaches explored distributed offloading techniques. For instance, DINA [14] employs a matching game approach for partitioning and offloading DNN tasks in a fog environment, aiming to minimize the total computation time. Another example is CoEdge [6], which operates as a distributed DNN computing system orchestrating collaborative DNN inference across diverse Edge devices.

B. THREE-TIER DNN PARTITIONING

DECC [8] dynamically divides DNNs into two or three parts and distributes these parts over Edge and Cloud, achieving the lowest latency. The emphasis is on maximizing throughput, and cost optimization is not addressed. In a previous study [13], a cost-driven offloading approach for DNNs with deadline constraints across Cloud, Edge, and IoT devices was introduced. The strategy employed a discrete Particle Swarm Optimization (PSO) to minimize system costs associated with executing DNN layers and data transfer. However, there's a risk of converging to a local optimum. Additionally, it did not take into account the variation in price/performance ratios for servers in each layer in real-world settings. Another PSO-based strategy is employed in a recent work [15] for vehicular Edge computing. The study only focuses on the tiny-YOLOv2 neural network which is a linear model. Authors in [11] introduced an adaptive framework for partitioning DNNs across end devices, Edge servers, and Cloud servers. The framework relies on layer prediction results to minimize DNN inference latency and does not consider cost optimization. Moreover, the algorithm's complexity makes it impractical for larger models. Many of these studies do not consider factors like propagation delay and the associated transmission and propagation costs, which are measured in terms of bandwidth expenses. This becomes especially crucial when contemplating a three-tier architecture. DDPQN [16] presents an improved Double Dueling Prioritized deep Q-Network algorithm aimed at achieving an efficient DNN offloading strategy with low delay, low energy consumption, and low cost in a local-edge-cloud collaborative environment. The algorithm typically requires up to 100 iterations to converge for simple models such as AlexNet and VGG. All the works discussed in this section involve examining every layer of DNNs, which is computationally intensive. In contrast, we uniquely identify specific split points, and this innovative approach will aid researchers in simplifying their partitioning strategies.

It's important to note that modifying the model is a method used to speed up the inference process in DNN models, but it often results in a decrease in model accuracy [17], [18]. Our approach to enhancing DNN inference speed achieves 100% accuracy as it does not involve altering the model. Hence, we refrain from making comparisons with strategies that do.

III. SYSTEM MODEL AND ASSUMPTIONS

We now define the system model by introducing crucial parameters and mathematical expressions pivotal to our analysis.

A. APPLICATION TASK MODEL

We calculate the computational requirements of each layer in different DNNs in terms of Floating Point Operations (FLOP) involved. Let AP_l^m represent the computational workload of layer l in the DNN application m, measured in GFLOP. We follow the sequential ordering of layers in a Directed

TABLE 1. Table for comparison of our splitting strategy against relevant research in the literature.

Research	[8]	[7]	[9]	[10]	[11]	[12]	This work
Architecture	2-tier (2 & 3 partitions)	2-tier	2-tier	2-tier	3-tier	2-tier	3-tier
Latency	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Cost	No	No	Yes	Yes	No	No	Yes
Throughput	Yes	No	No	No	No	Yes	Yes
Adaptive	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Complexity	$O((\#layers)^2)$	O(# layers)	O(# layers)	O(# layers)	$O((\#layers)^4)$	$> O((\#layers)^3)$	$<<$ O(# layers)

TABLE 2. Pricing of edge devices.

Cost analysis	Rpi 4 [20]	Jetson Nano [21]
Fixed cost		
Purchase cost	\$ 75	\$ 129
Cost per sec (lifetime 3 yrs)	$\$ 7.9 \times 10^{-7}$	$\$ 1.36 \times 10^{-6}$
Operational (OP) cost		
Maximum Power (P)	15 W	65 W
Electricity cost	\$ 0.15 per kWh	
OP cost (USD/sec)	$\$ 6.25 \times 10^{-5}$	$\$ 2.7 \times 10^{-4}$
Total cost K^p_{PE}	$\$ 6.32 \times 10^{-5}$	$\$ 2.71 \times 10^{-4}$

Acyclic Graph (DAG) in the order in which they are added. However, each Edge of a layer in the DAG structure is considered in the array of layer input. Let AD^m_l denote the input data size, measured in Mbit, needed for processing layer l of DNN m. The total computational demand for the entire application, denoted as AP^m, is the sum of individual layer demands AP^m_l for all layers l from 1 to L^m. Mathematically, it can be expressed as $AP^m = \sum_{l=1}^{L^m} AP^m_l$, where L^m represents the total number of layers in the application. Each application is bound by QoS requirements, with a specified deadline denoted by D^m, and budget constraints as per SLA, denoted by B^m.

B. THREE TIER COMPUTE AND PRICING MODEL

The computation capabilities of different computing entities are defined as P^p_E, P^q_{NE}, and P^r_C, measured in GFLOPS, for the Edge device p, NE instance q, and Cloud instance r respectively. For privately owned Edge devices, the cost encompasses both the purchase price and the electricity expenses for operating the device over a 3-year lifespan. The pricing details for Raspberry Pi 4 and Jetson Nano as Edge devices are presented in Table 2. For NE and Cloud, AWS instances [19] with on-demand pricing are considered.

Let K^p_{PE}, K^q_{PNE}, and K^r_{PC} denote the computation cost in \$/sec, of Edge device p, NE instance q, and Cloud instance r respectively. Let BW_{E-NE} and BW_{NE-C} denote the available bandwidth, measured in Mbps, to transmit data from Edge device to NE (E-NE) and NE to Cloud (NE-C), respectively. We use PD_{E-NE} and PD_{NE-C} to denote the propagation delay between E-NE and NE-C, respectively. Finally, let K^{BW}_{E-NE} and K^{BW}_{NE-C} denote the transmission cost over the network between E-NE and NE-C respectively, measured in USD/sec. Table 3 defines the notations used in this paper.

C. LATENCY AND COST OPTIMIZATION PROBLEM

Latency is defined as the cumulative duration of processing time, transmission time, and propagation times.

TABLE 3. Table of notations.

Model	Description
AP^m_l (GFLOP)	Processing needs of layer l of model m
AD^m_l (Mbit)	Input data size for processing layer l of m
AP^m (GFLOP)	Processing needs of entire model m
D^m (sec)	QoS deadline of model m
B^m (USD)	Model m's budget, as per the SLA
L^m	Number of layers of model m
Infrastructure	**Description**
P^p_E $(GFLOPS)$	Processing capacity of Edge device p
P^q_{NE} $(GFLOPS)$	Processing capacity of NE instance q
P^r_C $(GFLOPS)$	Processing capacity of Cloud instance r
K^p_{PE} (USD/sec)	Processing cost of Edge device p
K^q_{PNE} (USD/sec)	Processing cost of NE instance q
K^r_{PC} (USD/sec)	Processing cost of Cloud instance r
Network	**Description**
BW_{E-NE} (Mbps)	Bandwidth between Edge and NE
BW_{NE-C} (Mbps)	Bandwidth between NE and Cloud
PD_{E-NE} (sec)	Propagation delay between Edge and NE
PD_{NE-C} (sec)	Propagation delay between NE and Cloud
K^{BW}_{E-NE} (USD/sec)	Transmission cost between Edge and NE
K^{BW}_{NE-C} (USD/sec)	Transmission cost between NE and Cloud

1) LATENCY MODEL

We presume that the data originates from the Edge device, so we don't take into account any delay during data transmission when handling the DNN task at the Edge. Consequently, the latency at the Edge solely reflects the response time.

$$L^m_{Edge} = \frac{AP^m}{P^p_E} \tag{1}$$

The total latency at NE and Cloud also accounts for the transmission time and the propagation delay. Let T_{E-NE} and T_{E-C} be the functions to calculate the transmission time of input data from E-NE and Edge to Cloud respectively. T_{NE} and T_C are the functions to compute the processing time at NE and Cloud respectively.

$$T_{E-NE}(AD^m_l) = \frac{AD^m_l}{BW_{E-NE}} + PD_{E-NE} \tag{2}$$

$$T_{NE}(AP^m) = \frac{AP^m}{P^q_{NE}} \tag{3}$$

$$T_{NE-C}(AD^m_l) = \frac{AD^m_l}{BW_{NE-C}} + PD_{NE-C} \tag{4}$$

$$T_C(AP^m) = \frac{AP^m}{P^r_C} \tag{5}$$

Then the total latency at NE and Cloud are given by Eqs. 6 and 7 respectively.

$$L^m_{NE}(AP^m, AD^m_1) = T_{E-NE}(AD^m_1) + T_{NE}(AP^m) \quad (6)$$

$$L^m_{E-C}(AP^m, AD^m_1) = T_{E-NE}(AD^m_1) + T_{NE-C}(AD^m_1) + T_C(AP^m)$$

$$L^m_{NE-C}(AP^m, AD^m_1) = T_{NE-C}(AD^m_1) + T_C(AP^m) \quad (7)$$

2) COST MODEL

Next, we define the cost of computation of the application task at each tier. We consider the effect of utilization of different tiers in the cost computation similar to the approach outlined in Eq. 1 in the reference [22]. The formulas for calculating costs are provided as follows:

$$C^m_{Edge}(AP^m) = \frac{AP^m}{P^p_E} \times K^p_{PE} \quad (8)$$

$$C^m_{NE}(AP^m, AD^m_1) = K^{BW}_{E-NE} \times T_{E-NE}(AD^m_1) + K^q_{PNE} \times T_{NE}(AP^m) \quad (9)$$

$$C^m_{E-C}(AP^m, AD^m_1) = K^{BW}_{E-NE} \times T_{E-NE}(AD^m_1) + K^{BW}_{NE-C} \times T_{NE-C}(AD^m_1) + K^r_{PC} \times T_C(AP^m) \quad (10)$$

$$C^m_{NE-C}(AP^m, AD^m_1) = K^{BW}_{NE-C} \times T_{NE-C}(AD^m_1) + K^r_{PC} \times T_C(AP^m) \quad (11)$$

Next, we present the most important contribution of this work: finding the potential split points essential for any splitting strategy.

IV. LATENCY AND COST-EFFICIENT SPLIT POINT IDENTIFICATION

As earlier stated, not every DNN layer serves as an optimal split point; only a limited number of positions result in reduced latency or cost. In this section, we outline our most important contribution of identifying these viable split positions that help to reduce the complexity of any splitting strategy.

We assume the processing capacity of the tier increases as we move to the upper tier. ie, the Edge has a lower processing capacity compared to NE and NE has a smaller processing capacity compared to the Cloud.

$$P^p_E < P^q_{NE} < P^r_C \quad (12)$$

As a result, as we advance to the upper tiers, the processing time decreases due to increased processing capacity.

We assume that the processing cost in USD/GFLOP decreases as we move from the lower tier to the upper tier. The Edge has a higher processing cost for the same task than NE, and NE has a higher processing cost than the Cloud.

$$\frac{K^p_{PE}}{P^p_E} > \frac{K^q_{PNE}}{P^q_{NE}} > \frac{K^r_{PC}}{P^r_C} \quad (13)$$

As we advance to the right in the model's layers, the processing demand of the remaining model decreases. Therefore, only layers with lower transmission requirements than the previous layers can offer a latency/cost-efficient execution.

We assume that the bandwidth costs operate on a pay-as-you-use basis, wherein charges are incurred according to the bandwidth usage and are nonzero. Therefore we have,

$$K^{BW}_{E-NE} > 0; \quad K^{BW}_{NE-C} > 0 \quad (14)$$

Let SP^m be the set of possible split points for a model m with L^m layers. We start looping over the layers from 1 to L^m to find the latency and cost-efficient split points. The following equation describes how the set SP^m is updated:

$$SP^m = \begin{cases} \{1\}, & \text{if } |SP^m| = 0 \\ SP^m \cup \{i\}, & \text{if } \mathrm{AD}_i < \mathrm{AD}_{\max(SP^m)} \text{ for } i \in [2, L^m] \end{cases} \quad (15)$$

where $|SP^m|$ represents the cardinality of set SP^m. The possible split points do not depend on the bandwidth values between the different tiers.

A. PROOFS FOR LATENCY AND COST EFFICIENCY OF CHOSEN SPLIT POINTS

In this subsection, we illustrate the advantages of these split points for latency and cost-efficient partitioning and demonstrate that considering any layer not in SP^m as a split point would result in higher latency and costs.

Let APF^m_{s1-1} denote the sum of the processing requirements of the model from layer 1 to the split point $s1-1$. It is computed by summing up the computations of all the layers, given as:

$$APF^m_{s1-1} = \sum_{l=1}^{s1-1} AP^m_l; \quad \forall s1 \in SP^m$$

Latency Efficiency of Proposed Split Points:

The latency of processing the entire model at the NE will be represented as:

$$\frac{AD_1}{BW_{E-NE}} + PD_{E-NE} + \frac{AP^m}{P^q_{NE}}$$

Now if initial $s1-1$ layers are offloaded on the Edge while the rest are being processed at the NE, then we have the processing time as:

$$\frac{APF^m_{s1-1}}{P^p_E} + \frac{AD_{s1}}{BW_{E-NE}} + PD_{E-NE} + \frac{AP^m - APF^m_{s1-1}}{P^q_{NE}}$$

Now the partitioning as layer $s1$ will be preferred only if it provides a lower latency which means

$$\frac{APF^m_{s1-1}}{P^p_E} + \frac{AD_{s1}}{BW_{E-NE}} + PD_{E-NE} + \frac{AP^m - APF^m_{s1-1}}{P^q_{NE}} < \frac{AD_1}{BW_{E-NE}} + PD_{E-NE} + \frac{AP^m}{P^q_{NE}}$$

which on further simplification gives:

$$APF^{m}_{s1-1} \cdot (\frac{1}{P^{p}_{E}} - \frac{1}{P^{q}_{NE}}) < \frac{AD_1 - AD_{s1}}{BW_{E-NE}}$$

From Eq. 12, we observe that the left-hand side is positive. Therefore, the right-hand side should also be positive for the inequality to hold. This implies $AD_1 > AD_{s1}$, indicating that the amount of data to be transmitted at any subsequent split point must be lower than that at any previously considered split positions.

Cost Efficiency of Proposed Split Points:

The cost of execution of the entire model, with processing requirement AP^m, at NE is expressed as:

$$K^{BW}_{E-NE} \times (\frac{AD_1}{BW_{E-NE}} + PD_{E-NE}) + K^{q}_{PNE} \times \frac{AP^m}{P^{q}_{NE}}$$

The cost of model execution, when divided into two partitions, with the initial partition running at the Edge and the second partition executed at NE, is expressed as follows:

$$K^{p}_{PE} \times \frac{APF^{m}_{s1-1}}{P^{p}_{E}} + K^{q}_{PNE} \times \frac{(AP^m - APF^{m}_{s1-1})}{P^{q}_{NE}} + K^{BW}_{E-NE} \times (\frac{AD_{s1}}{BW_{E-NE}} + PD_{E-NE})$$

Let's compare the cost of executing the entire model on NE to the cost of offloading the initial $s1-1$ layers on the Edge and executing the remaining on the NE to ensure a smaller cost in the latter. The following conditions must be satisfied:

$$APF^{m}_{s1-1} \cdot (\frac{K^{p}_{PE}}{P^{p}_{E}} - \frac{K^{q}_{PNE}}{P^{q}_{NE}}) < K^{BW}_{E-NE} \times (\frac{AD^{m}_{1} - AD^{m}_{s1}}{BW_{E-NE}})$$

As per Eq 13, given that the left side is positive, it implies that the expression on the right side must also be positive. Also according to Eq. 14 bandwidth cost is non-zero. Consequently, this results in the inequality $AD^{m}_{1} > AD^{m}{}_{s1}$, and this inequality must hold for each subsequent split. In other words, the data transmission requirement at each new split position is the minimum among the data transmissions at all previous split positions.

For both latency and cost improvement, we have demonstrated that only the split points detected by Eq. 15 can serve as viable split points, and no other layer can result in lower latency or cost.

B. IDENTIFICATION OF SPLIT POINTS FOR VARIOUS DNN MODELS

In this section, we apply the strategy introduced in the previous section to identify the set of potential split points for nine different DNN models. Table4 presents the split points detected by Eq. 15 for these models. It is crucial to note that, based on the evidence presented above, we can confidently assert that the split points listed in the table are the only feasible points that may result in latency and cost reduction.

Furthermore, Figure 1 illustrates the reduction in complexity achieved by considering only the split points shown in Table 4 compared to considering all layers of the models.

TABLE 4. Model split points.

Model Name	Split Points
VGG16	[1, 15, 19, 21]
MobileNetV2	[1, 17, 35, 62, 124, 155]
ResNet50	[1, 40, 82, 144, 176]
EfficientNetB0	[1, 12, 14, 28, 241]
InceptionV3	[1, 18, 19, 23, 35, 96, 243, 244, 312]
ResNet101	[1, 40, 82, 314, 346]
ResNet152	[1, 40, 122, 484, 516]
DenseNet169	[1, 53, 141, 144, 372, 596]
InceptionResNetV2	[1, 18, 19, 23, 42, 276, 610, 611, 619, 781]

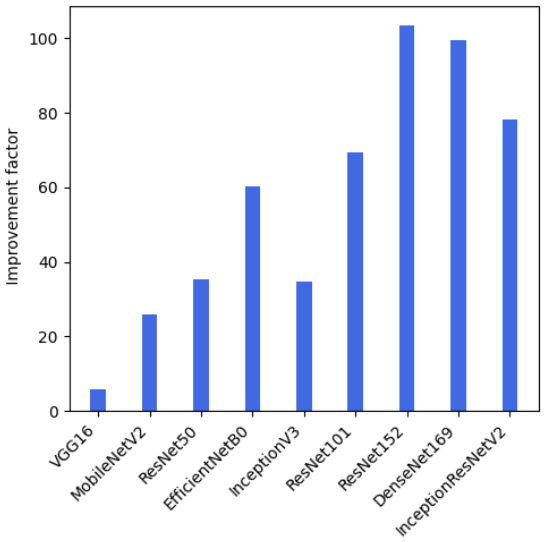


FIGURE 1. Complexity reduction in searching for possible split point for different DNN models sorted in the increasing order of the number of layers.

The models on the X-axis are ordered in increasing order of the number of layers, and the Y-axis shows the improvement factor in terms of the layers that need to be evaluated to find a latency or cost-efficient split point. The figure demonstrates that for larger models, the improvement factor can be as high as 100 times. We consider the example of the ResNet50 model, which consists of 177 layers. However, not all these layers act as potential split points, and only 5 layers are potential split positions as determined by Eq. 15. Figure 2 illustrates how these 5 potential partition points for the ResNet 50 model helps to achieve a cost reduction. On the x-axis, we present the layer names and corresponding layer numbers of the possible split points, while the y-axis represents the associated cost in USD. Within each split position, the first bar represents the processing cost, and the second bar represents the transmission cost. The blue portion of the first bar shows the cost of Edge processing, and the orange part shows the sum of transmission and processing costs at NE. In the first scenario, when the entire model is processed at the NE, the processing cost is relatively low compared to the second scenario in which 39 layers are processed at the Edge, with the remaining layers processed at NE. This observation highlights that with an increase in Edge processing the overall processing cost increases. The same holds for latency reduction. Therefore, to achieve a lower

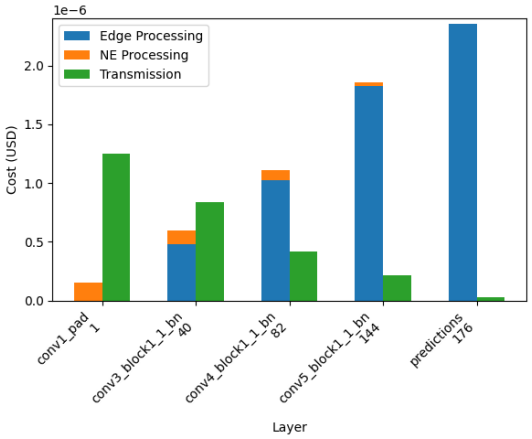


FIGURE 2. Processing and transmission cost at each possible split point for ResNet50 model with 177 layers.

total cost/latency, the increase in processing cost/time must be compensated by a corresponding reduction in transmission cost/time, resulting in a lower total cost/latency than that of the previous configuration. Consequently, a partition point is latency or cost-efficient only if the transmission requirements at that point are lower than those of the previous partition, where a greater portion of processing occurred at the upper layers.

C. GENERALIZING SPLIT POINTS FOR ANY NUMBER OF TIERS

These identified split points remain applicable for any number of tiers as long as the conditions outlined in Eqs 12, 13 and 14 are satisfied across the multi-tier architecture. Let k be the number of tiers in which the DNN has to be split. Then, the complexity of adapting the split decision with any variation in infrastructure or bandwidths would be $\mathcal{O}(|SP^m|^k)$.

It's worth noting that in certain scenarios, the DNN model output is needed on the Edge tier. We have not considered this case in our strategy. However, even in such scenarios, the complexity remains unchanged. In this case, either the final output is sent to the Edge, or if the final output is larger than the last split point, then the last split point will always be chosen as the fourth split point. This is because the last split point always results in the least processing being sent to the lower tier, where both the processing time and cost increase according to Eqs 12, 13 and 14.

V. OPTIMAL PLACEMENT ALGORITHMS

In this section, we formulate the two problems outlined in Subsection I-A and present the algorithms necessary to address these optimization challenges.

A. DNN PARTITIONING PROBLEM

In this subsection, we define the problem of partitioning a DNN application into segments for execution in different tiers to minimize latency and cost compared to single-tier execution. Let $s1$ and $s2$ represent the two split positions for model m. In this arrangement, the segment preceding layer $s1-1$ is processed at the Edge, the input to layer $s1$ is transmitted to the NE, and layers $s1$ to $s2-1$ are executed at the NE. Subsequently, the input to layer $s2$ is sent to the Cloud, and the remaining part is processed in the Cloud. Consequently, the overall latency and cost for the DNN model's execution can be calculated using Eqs 16 and 17, respectively.

$$\begin{aligned} L_{split}(s1,s2) &= L_{Edge}(APF^m_{s1-1}) \\ &+ L_{NE}((APF^m_{s2-1} - APF^m_{s1-1}), AD^m_{s1}) \\ &+ L_{NE-C}((AP^m - APF^m_{s2-1}), AD^m_{s2}) \end{aligned} \tag{16}$$

$$\begin{aligned} C_{split}(s1,s2) &= C_{Edge}(APF^m_{s1-1}) \\ &+ C_{NE-C}((APF^m_{s2-1} - APF^m_{s1-1}), AD^m_{s2}) \\ &+ C_{NE}((AP^m - APF^m_{s2-1}), AD^m_{s1}) \end{aligned} \tag{17}$$

Our objective is to find the optimal split, that is, the layer $s1$ and $s2$ for which the two objectives defined in Sec. I-A are achieved while satisfying the QoS aligned with the model's deadline and adhering to budget constraints defined by the SLA. The problems of latency and cost minimization are represented by Equations 18 and 19, respectively.

$$\begin{aligned} L^m_{OP} = \arg\min_{s1,s2} \quad & L_{split}(s1,s2) \\ \text{s.t. } & L^m_{OP} < D^m \\ & C^m_{OP} < B^m \\ & 1 \le s1 \le s2 \le L^m \end{aligned} \tag{18}$$

$$\begin{aligned} C^m_{OP} = \arg\min_{s1,s2} \quad & C_{split}(s1,s2) \\ \text{s.t. } & L^m_{OP} < D^m \\ & C^m_{OP} < B^m \\ & 1 \le s1 \le s2 \le L^m \end{aligned} \tag{19}$$

A naive approach to finding two optimal split positions results in an algorithmic complexity of $\mathcal{O}(|SP^m|^2)$. However, we reduce our algorithm's complexity to linear by addressing the two problems defined in Sec.I separately. First, we find the tier providing the lowest latency for the single-tier execution of the entire model. Subsequently, Algorithms 2, 3, and 4 solve the problem of finding the optimal split positions in three different scenarios. From here onwards, we only provide steps for solving the problem in Eq.18 due to space constraints, and a similar approach is applied for problem defined in Eqs.19. For the minimum latency solution, we find the optimal tier that provides the minimum latency while satisfying the SLA constraints. If no solution exists that satisfies SLA and QoS, we pick the solution that provides minimum latency as our objective is latency minimization. The steps to solve the single-tier problem are outlined in Algorithm1.

In the first scenario, where Edge execution offers the lowest latency, Algorithm 2 begins by iterating through potential split points across the model layers. At each split point si, the algorithm assesses the time required to compute

Algorithm 1 Optimal placement that yields minimum latency

1: **Input:** edge, ne, cloud, model, and network params
2: **Output:** OP(L,C,s1,s2)
3: Compute latency and costs using functions in Eq. 1 and 6
4: Sort $L^m_{Edge}, L^m_{NE}, L^m_{E-C}$ in increasing order
5: Choose L^m_{OP} that satisfies QoS and SLA constraints
6: **if** No solution exists that satisfies both Qos and SLA constraints **then**
7: Choose the solution that provides minimum latency.
8: **end if**
9: **if** Optimal tier == Edge **then**
10: OP($L_{Edge}, C_{Edge}, L^m, L^m$)
11: OP = Algorithm 2
12: **else if** Optimal tier == NE **then**
13: OP($L^m_{NE}, C^m_{NE}, 0, L^m$)
14: OP = Algorithm 3
15: **else**
16: OP($L^m_{E-C}, C^m_{E-C}, 0, 0$)
17: OP = Algorithm 4
18: **end if**
19: **return** OP

Algorithm 2 Function when Edge is optimal

1: **Input:** edge, ne, cloud, model, and network params
2: **Output:** OP(L,C,s1,s2)
3: **for** $si \in SP^m$ **do**
4: **if** $L_{split}(si, si) < L_{split}(si, L^m)$ **then**
5: **if** $L_{split}(si, si) < L^m_{OP}$ & $C_{split}(si, si) < B^m$ **then**
6: OP = $(L_{split}(si, si), C_{split}(si, si), si, si)$
7: **end if**
8: **end if**
9: **end for**
10: **for** $si \in SP^m$ & si $\leq$ s2 **do**
11: **if** $L_{split}(si, s2) < L^m_{OP}$ & $C_{split}(si, s2) < B^m$ **then**
12: OP=$(L_{split}(si, s2), C_{split}(si, s2), si, s2)$
13: **end if**
14: **end for**
15: **return** OP

Algorithm 3 Function when NE is optimal

1: **Input:** edge, ne, cloud, model, and network parameters
2: **Output:** OP(L,C,s1,s2)
3: **for** $si \in SP^m$ **do**
4: **if** $L_{split}(OP.s1, si) < L^m_{OP}$ & $C_{split}(s1, i) < B^m$ **then**
5: OP= $L_{split}(s1, si), C_{split}(s1, si), s1, si)$
6: **end if**
7: **end for**
8: **for** $si \in SP^m$ & si $\leq$ s2 **do**
9: **if** $L_{split}(si, s2) < L^m_{OP}$ & $C_{split}(si, s2) < B^m$ **then**
10: OP=$L_{split}(si, s2), C_{split}(si, s2), si, s2)$
11: **end if**
12: **end for**
13: **return** OP

Algorithm 4 Function when Cloud is optimal

1: **Input:** edge, ne, cloud, model, and network parameters
2: **Output:** OP(L,C,s1,s2)
3: **for** $si \in SP^m$ **do**
4: **if** $L_{split}(si, si) < L_{split}(0, si)$ **then**
5: **if** $L_{split}(si, si) < L^m_{OP}$ & $C_{split}(si, si) < B^m$ **then**
6: OP = $(L_{split}(si, si), C_{split}(si, si), si, si)$
7: **end if**
8: **end if**
9: **end for**
10: **for** $si \in SP^m$ & v $\geq$ s1 **do**
11: **if** $L_{split}(s1, si) < L^m_{OP}$ & $C_{split}(s1, si) < B^m$ **then**
12: OP=$L_{split}(s1, si), C_{split}(s1, si), s1, si)$
13: **end if**
14: **end for**
15: **return** OP

the model from layer $si + 1$ to the last layer at both Cloud and NE in line 4. If the computation time is shorter on the Cloud, the algorithm checks in the next line if executing this part on the Cloud also provides a lower latency compared to the current solution (i.e., Edge execution) (line 5). If this condition holds, it implies that the selected partition yields the lowest latency when executed on the Cloud, and so the algorithm allocates this partition to the Cloud and designates this split point as $s2$; otherwise, $s2$ retains its original value which is L^m as carried over from line 10 of Algorithm 1. In both cases, the algorithm then evaluates whether allocating a portion to the NE can further reduce latency (lines 10 to 14). If any partition from split point si until the part of the model that has already been offloaded to the Cloud achieves a lower latency by offloading the part to NE, which means out of the three tiers NE provides the lowest latency to execute this part, then we allocate that part to NE. In cases where offloading does not lead to reduced latency, the algorithm converges to the Edge-only solution. This optimization simplifies the *DNNSplit* algorithm, reducing the search for two split positions to $\mathcal{O}(2 \cdot |SP^m|)$.

Similarly, when the NE execution achieves the minimum latency, we first investigate the later segments, extending from each split point si to the model's last layer, intending to allocate them for Cloud execution. As we already know the Edge cannot yield lower latency for the later portions. Hence in Algorithm 3, in lines 3 to 7, we examine whether a segment can decrease latency when allocated for Cloud execution. If such a segment is identified, it is allocated to the Cloud, and the split point is denoted as $s2$. Subsequently, from the first layer up to layer $s2$ (lines 8 to 11), we determine whether any section can achieve lower latency through Edge execution; if affirmative, we assign that section to the Edge owing to its superior performance and allocate the split point as $s1$. This decision is based on our prior knowledge that NE provides lower latency for the starting layers as compared

to the Cloud since the single-tier execution yields the lowest latency at NE.

In scenarios where the Cloud yields the overall minimum execution time for the entire model, Algorithm 4 identifies the split points that offer lower latency compared to single-tier execution. The algorithm first examines whether a partition from layer 1 to a specific split point si achieves the shortest execution time on the Edge (as seen in lines 5 and 6). Following this evaluation, this partition is allocated to the Edge, and both split points are assigned to this split point si, denoting that the model is being executed between the Edge and the Cloud. In case no partition, when offloaded to the Edge, achieves a lower latency, the values of $s1$ and $s2$ remain unchanged as carried over from line 16 in Algorithm 1. For the remaining segment from $s1$ to the final layer, the algorithm (in lines 10 to 15) checks whether any partition can be processed with lower latency on the NE. Subsequently, it allocates this partition to the NE, designating this split point as $s2$. In all three scenarios, the identification of the two split points, $s1$ and $s2$, involves looping over all possible split points twice, resulting in a total time complexity of $\mathcal{O}(2 \cdot |SP^m|)$.

VI. EFFICIENCY OF DNNSPLIT IN THROUGHPUT IMPROVEMENT

In this section, we define the throughput achieved by processing the DNN model across different tiers. Let TH_{Edge}, TH_{NE}, and TH_{Cloud} represent the functions calculating the maximum requests processed per second at the Edge, NE, and Cloud, respectively. The number of requests that can be processed at NE will depend on the throughput achieved by the transmission link between Edge and NE and the processing throughput at the NE. The maximum throughput will be determined by the slowest processing segment. The overall throughput will be the minimum of the two, as given in Eq.21. Similarly, the throughput of the Cloud execution, as given in Eq.23, accounts for both the transmission and processing throughput.

$$TH^m_{Edge}(AP^m) = \frac{1}{L_{Edge}(AP^m)} \tag{20}$$

$$TH^m_{NE}(AP^m, AD^m_1) = min\left(\frac{1}{T_{E-NE}(AD^m_1)}, \frac{1}{T_{NE}(AP^m)}\right) \tag{21}$$

$$TH^m_{E-C}(AP^m, AD^m_1) = min\left(\frac{1}{T_{E-NE}(AD^m_1) + T_{NE-C}(AD^m_1)}, \frac{1}{T_C(AP^m)}\right)$$

$$TH^m_{NE-C}(AP^m, AD^m_1) = min\left(\frac{1}{T_{NE-C}(AD^m_1)}, \frac{1}{T_C(AP^m)}\right) \tag{22}$$

Let $s1$ and $s2$ represent two latency and cost-efficient split points computed by Eq. 15, the throughput achievable by partitioning the model into three tiers by using these split points is defined in Eq. 23.

$$TH_{split}(s1, s2) = \min\Big(TH_{Edge}(APF^m_{s1}), TH_{NE}((APF^m_{s2-1} - APF^m_{s1-1}), AD^m_{s1}), TH_{NE-C}((AP^m - APF^m_{s2-1}), AD^m_{s2})\Big) \tag{23}$$

Further, the throughput improvement problem is defined in Eq. 24. It is important to note that we exclusively utilize the latency and cost-efficient split points to enhance throughput while meeting QoS and SLA requirements. This does not represent the maximum throughput of the system but rather the maximum throughput improvement attainable with latency and cost-efficient split points while adhering to the constraints.

$$TH^m_{OP} = \arg\max_{s1,s2} TH_{split}(s1, s2) \quad \text{s.t. } L^m_{OP} < D^m,\ C^m_{OP} < B^m,\ 1 \le s1 \le s2 \le L^m \tag{24}$$

We employ a two-step strategy to address the throughput improvement problem, akin to our approach for latency and cost minimization. Initially, we identify the tier with the highest throughput while ensuring compliance with latency and cost constraints. This involves selecting the maximum throughput from among the three equations: Equations 20, 21, and 22. If a viable solution exists, it is selected. Subsequently, based on the initially selected tier, we employ Algorithm 5 if the Edge tier is chosen, Algorithm 6 if the NE tier is selected, and Algorithm 7 if the Cloud tier is opted for in the initial step. In each of these situations, determining the two split points, $s1$ and $s2$, requires iterating over all potential split points twice, leading to a total time complexity of $\mathcal{O}(2 \cdot |SP^m|)$. Alternatively, if no single-tier execution satisfies

Algorithm 5 Function when Edge is optimal

1: **Input:** edge, ne, cloud, model, and network params
2: **Output:** OP(L,C,TH,s1,s2)
3: **for** $si \in SP^m$ **do**
4: **if** $TH_{split}(si, si) > TH_{split}(si, L^m)$ **then**
5: **if** $TH_{split}(si, si) > TH^m_{OP}$ & $C_{split}(si, si) < B^m$ & $L_{split}(si, si) < D^m$ **then**
6: OP = $(L_{split}(si, si), C_{split}(si, si), TH_{split}(si, si), si, si)$
7: **end if**
8: **end if**
9: **end for**
10: **for** $si \in SP^m$ & si $\le$ s2 **do**
11: **if** $TH_{split}(si, s2) > TH^m_{OP}$ & $C_{split}(si, s2) < B^m$ & $L_{split}(si, si) < D^m$ **then**
12: OP=$(L_{split}(si, s2), C_{split}(si, s2), TH_{split}(si, s2), si, s2)$
13: **end if**
14: **end for**
15: **return** OP

both QoS and SLA constraints, we prioritize solutions that meet either of the constraints. In situations where neither constraint can be fulfilled, we select the solution with the minimum latency.

Algorithm 6 Function when NE is optimal

Input: edge, ne, cloud, model, and network parameters
Output: OP(L,C,MT,s1,s2)
for $si \in SP^m$ **do**
  if $TH_{split}(s1, si) > TH^m_{OP}$ & $C_{split}(s1, i) < B^m$ & $L_{split}(si, si) < D^m$ **then**
    OP= $L_{split}(s1, si)$, $C_{split}(s1, si)$, TH_{split} $(s1, si)$, $s1$, si)
  end if
end for
for $si \in SP^m$ & si $\leq$ s2 **do**
  if $TH_{split}(si, s2) > TH^m_{OP}$ & $C_{split}(si, s2) < B^m$ & $L_{split}(si, si) < D^m$ **then**
    OP=($L_{split}(si, s2)$, $C_{split}(si, s2)$, TH_{split} $(si, s2)$, si, $s2$)
  end if
end for
return OP

Algorithm 7 Function when Cloud is optimal

Input: edge, ne, cloud, model, and network parameters
Output: OP(L,C,TH,s1,s2)
for $si \in SP^m$ **do**
  if $TH_{split}(si, si) > TH_{split}(0, si)$ **then**
    if $TH_{split}(si, si) > TH^m_{OP}$ & $C_{split}(si, si) < B^m$ & $L_{split}(si, si) < D^m$ **then**
      OP = ($L_{split}(si, si)$, $C_{split}(si, si)$, $TH_{split}(si, si)$, si, si)
    end if
  end if
end for
for $si \in SP^m$ & v $\geq$ s1 **do**
  if $TH_{split}(s1, si) > TH^m_{OP}$ & $C_{split}(s1, si) < B^m$ & $L_{split}(si, si) < D^m$ **then**
    OP=$L_{split}(s1, si)$, $C_{split}(s1, si)$, $TH_{split}(s1, si)$, $s1$, si)
  end if
end for
return OP

VII. EVALUATION

In this section, we present the experimental results of our algorithm. Our investigations involve two Edge devices, Rpi and Jetson Nano, with their specifications detailed in Table 2. At NE, we consider a GPU device, the specifications of which are presented in Table 5. It's important to note that the listed processing speed for the GPU instance reflects its peak performance when all cores are actively involved in executing a task. We treat this as a customizable service and experiment with various combinations of processor sharing. Specifically, we examine scenarios with 1 and 5 virtual GPUs (vGPUs) at the NE and in the Cloud, respectively. The processing ratio of NE to Rpi is 4333, while the NE to Jetson Nano ratio is 650. Unlike prior research, we account for the propagation delay between distinct tiers, setting the propagation delay between Edge and NE to 1 ms in adherence to [3]. Table 6 outlines the constant parameters that we used in all our experiments. The transmission cost is calculated using the monthly internet expenses of the AT&T network which is considered to be $20 per month for a maximum of 300 Mbps. We conduct experiments on nine benchmark models implemented within the tensorflow keras module [23], and the specific details of these models are provided in Table 7 in the increasing order of their computational requirement.

TABLE 5. Description of AWS instance.

AWS instance	Hardware	Processing GFLOPS	On Demand Price USD/sec
GPU	p4d.24xlarge	312000	$0.0011378

TABLE 6. Description of parameter settings.

PD_{E-NE}	PD_{NE-C}	K^{BW} $/mon	K^{BW} $/sec
0.001 *sec*	0.05 *sec*	$ 20 for 300 Mbps	$ 7.716×10^{-6}

TABLE 7. Model information.

Model Name	Model Proc (GFLOP)	# layers	Complexity
MobileNetV2	0.98076004	156	2×6
EfficientNetB0	2.039162056	241	2×4
DenseNet169	3.553713448	597	2×6
ResNet50	4.095510375	177	2×5
ResNet101	7.819725672	347	2×5
InceptionV3	10.0067186	313	2×9
ResNet152	11.54674468	517	2×5
VGG16	15.476394984	23	2×4
InceptionResNetV2	20.262857576	782	2×10

We conduct a comparative analysis by benchmarking our strategy against three single-tier execution strategies: Edge-only, NE-only, and Cloud-only. For single-tier latency computations, we use the formulas in Eqs.1,6, and 7. For single-tier cost computations, we use Eqs. 8, 9, and, 10. In our evaluation, we utilize the SpeedUp metric for comparative analysis, expressing it as the ratio of the execution time on a single tier to the execution time achieved by the *DNNSplit* algorithm. We measure cost reduction through the Cost Reduction Factor(CRF), calculated as the ratio of the single-tier execution cost to the cost achieved by the *DNNSplit* algorithm. A SpeedUp/CRF of 1 indicates no latency/cost reduction, while a SpeedUp/CRF greater than 1 signifies a reduction in latency/cost.

A. EVALUATION OF SPLIT POINT ROBUSTNESS: DETECTED POINTS VS. ALL LAYERS

In this experiment, we assess the robustness of split points by comparing the outcomes when considering all layers of the models versus only the layers detected by Eq.15 as shown in Table 4. Figure 3 compares latency speedup

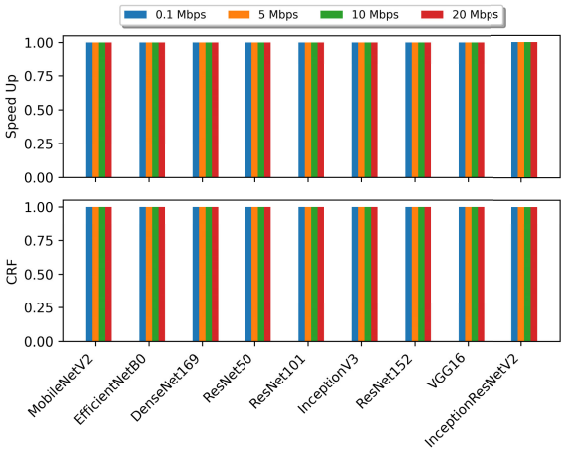


FIGURE 3. Speedup and CRF values achieved by *DNNSplit* for different models as compared to considering all the layers of the models.

and CRF for both strategies. Remarkably, both strategies yield identical results with a speedup and CRF of 1 across various bandwidths, indicating consistent latency and cost outcomes. We specifically selected lower bandwidths for this experiment because, at these levels, the partitioning decisions are more prone to change.

Table 7 summarizes the computational complexity involved in determining latency and cost-efficient partitioning for various models. It demonstrates the reduction in algorithmic complexity when adjusting split decisions with network variations.

B. LATENCY AND COST REDUCTION COMPARED TO EDGE TIER

In this experiment, we compare our partitioning strategy in terms of speedup and cost reduction achieved compared to Edge-only execution for two Edge devices. We use a low-bandwidth scenario in this case because the Edge-only solution has the potential to achieve the lowest latency and cost when the bandwidth is small. We establish the QoS requirement to be speedup $= 1$ for each model and set the SLA constraint to achieve CRF $= 1$ for both Edge devices.

The results of our algorithm are illustrated in Figure 4. On the X-axis, we have various DNN models arranged in ascending order of their computational requirements, and the Y-axis depicts speedup values in the upper plot and CRF in the lower plot. The black dotted line indicates speedup $= 1$ in the upper plot and CRF $= 1$ in the lower plot.

The graph shows that both Rpi and Jetson exhibit an increase in speedup and CRF value as bandwidth increases. This observation indicates that as bandwidth increases, the *DNNSplit* algorithm dynamically shifts more layers to the NE or Cloud, effectively adapting to the bandwidth increment. Additionally, the figure illustrates that *DNNSplit* successfully meets both QoS and SLA constraints across all bandwidth scenarios and DNN models considered. Furthermore, it demonstrates that due to the lower processing capacity of the RPi device, none of the models opt for Edge-only execution, even at low bandwidths. Conversely, when Jetson is utilized as the Edge device, Edge-only execution is favored in most cases due to its lower latency and cost-effective performance.

Note that the EfficientNetB0 model achieves exceptionally high speedup and CRF values in both scenarios. Therefore, it has been removed from the figure for clarity. Specifically, when RPi is used as the Edge device, it achieves a speedup of up to 40x and a CRF of up to 30x. Conversely, when Jetson is employed at the Edge, a speedup of 20x and a CRF of 15x are attained.

Our second experiment demonstrates the latency and cost-reducing effectiveness of *DNNSplit* compared to NE-only and Cloud-only scenarios. We fixed the Edge device to Rpi and evaluated the performance of *DNNSplit* with varying network bandwidths between E-NE and NE-C. The goal is to highlight how our algorithm dynamically adjusts partitioning decisions in response to changes in bandwidth across the three tiers to achieve lower latency and cost compared to single-tier execution.

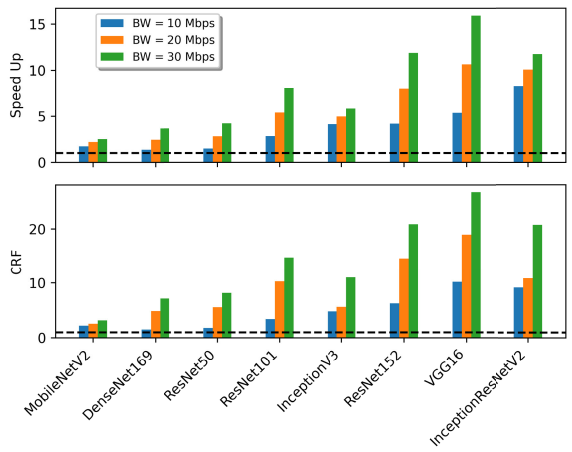
We consider processing at the Cloud to be 25 times cheaper than at the NE. The QoS and SLA constraints are the same as in the previous experiment. Figure 5 visually demonstrates the achieved reduction in latency and costs by the *DNNSplit* algorithm while satisfying both QoS and SLA constraints.

Figure 5a illustrates the latency and cost reductions compared to NE-only execution under three different bandwidth conditions. The figure demonstrates that *DNNSplit* achieves a speedup of up to 4x and a CRF improvement of up to 1.5x compared to NE-only execution. Similarly, Figure 5b presents the results compared to Cloud-only execution. *DNNSplit* achieves up to 5x speedup and 2x CRF enhancement.

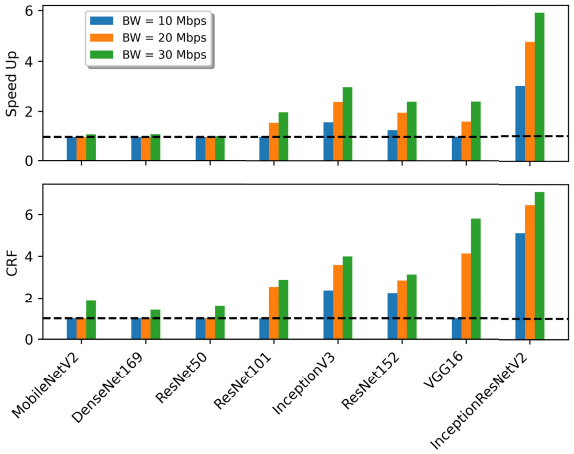
It's worth noting that the model EfficientNetB0 has been omitted for clarity from Figure 5a, where it achieves a speedup of up to 30x and a cost reduction of up to 8x. As depicted in Figure 5b, this model attains up to 15x speedup and 4x cost reduction compared to Cloud-only execution.

C. THROUGHPUT IMPROVEMENT

In this section, we've incorporated our throughput improvement algorithm presented in Section VI to highlight how *DNNSplit* significantly improves throughput. By segmenting the model into distinct tiers and concurrently executing and transmitting different parts, our strategy showcases noteworthy results. We show the throughput improvement with Rpi as the Edge device and 30 Mbps as the bandwidth between the three tiers. As illustrated in Figure 6, itDNNSplit attains a remarkable increase in throughput, reaching up to 5 times higher compared to that achieved by running the entire model in a single tier. Importantly, this improvement is achieved while adhering to the model's specified deadline and budget constraints. Note that smaller models, such as MobileNetV2, EfficientNetB0, and DenseNet169, have been excluded from the figure. These models exhibit a preference for Edge-only execution at lower bandwidths, resulting in throughput equal to Edge-only throughput.

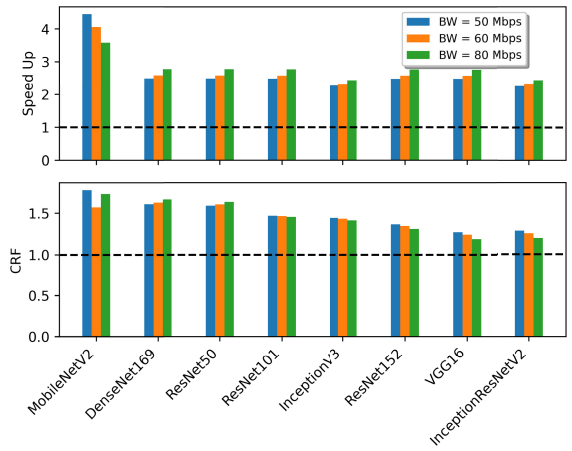


(a) Latency and cost reduction compared to Edge-only execution for different models when Rpi is used as an Edge Device

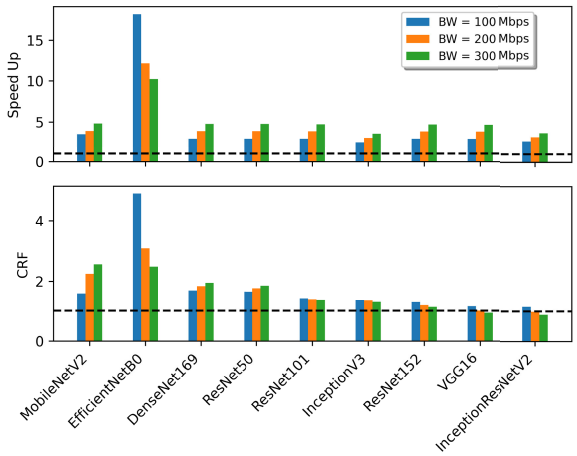


(b) Latency and cost reduction compared to Edge-only execution for different models when Jetson Nano is used as an Edge Device

FIGURE 4. Results of our splitting algorithm for different models showing how the algorithm adjusts its partitioning decision, for minimizing the total latency and cost, with varying infrastructure and bandwidth scenario.



(a) Latency and cost reduction for different models compared to NE-only execution with different bandwidths



(b) Latency and cost reduction for different models compared to Cloud-only execution with different bandwidths

FIGURE 5. Results of our splitting algorithm for different models showing how efficient the algorithm is in achieving speedup and cost reduction with varying bandwidths.

D. IMPACT OF BANDWIDTHS AND TRANSMISSIONS COSTS

In this section, we conducted experiments to evaluate the impact of varying combinations of transmission bandwidths and transmission costs between Edge-to-NE and NE-to-Cloud. Throughout this experiment, the Edge device was fixed as Rpi, and the NE was set to 1vGPU. We assumed that the computational cost at the Cloud is 25 times more economical than that at the NE. We are exploring four potential bandwidth options (1, 10, 100, and 1000 Mbps) between E-NE and NE-C. Moreover, we examined three distinct factors for the bandwidth cost: $a \times K^{BW}$ for K^{BW}_{E-NE} and $b \times K^{BW}$ for K^{BW}_{NE-C}, where a and b can assume values from the set $\{0.1, 1, 10\}$. The combinations of bandwidths and costs are represented as tuples with four values: $[BW_{E-NE}, a, BW_{NE-C}, b]$. Table 8 displays models that exhibit a preference for computation entirely on the Edge, NE, or a partition between the Edge and NE to achieve minimum cost, considering all considered bandwidth and cost values. Table 9 illustrates the model VGG16, which does not favor partitioning. Lastly, Table 10 indicates that larger models prefer three-tier partitioning at higher bandwidths and lower costs from NE to Cloud.

E. KEY FINDINGS

The algorithm dynamically adjusts its partitioning decisions based on changes in infrastructure scenarios, demonstrating its capability to respond to varying cost ratios, bandwidth conditions, and device capabilities. The variations in the best partition point suggest that there is a need for a system to partition DNN computation between the 3-tiers.

- **Edge Device Impact:** The selection of an Edge device (RPi vs. Jetson Nano) significantly influences the partitioning decision. The Jetson device, with superior computational capacity, tends to handle more computational load. This effect is particularly pronounced in

TABLE 8. Models that achieve minimal costs through single or two-tier partitioning.

Condition	MobileNetV2	EfficientNetB0	DenseNet169	ResNet50	ResNet101	InceptionV3
[1, ≤ 1,Any,Any]	Edge-only	[14,241,0]	Edge-only	Edge-only	Edge-only	[96,313,0]
[1, 10 ,Any,Any]	Edge-only	[14,241,0]	Edge-only	Edge-only	Edge-only	Edge-only
[10, ≤ 1, Any,Any]	[62,156,0]	[14,241,0]	[144,597,0]	[82,177,0]	[82,347,0]	[35,313,0]
[10, 10, Any,Any]	[62,156,0]	[14,241,0]	[144,597,0]	[144,177,0]	[314,347,0]	[96,313,0]
[10, ≥ 1, Any,Any]	Edge-only	[14,241,0]	Edge-only	Edge-only	Edge-only	Edge-only
[100, 0.1 ,Any,Any]	[17,156,0]	[14,241,0]	NE-only	NE-only	NE-only	NE-only
[100, 1,Any,Any]	Edge-only	[14,241,0]	Edge-only	Edge-only	[82,347,0]	[35,313,0]
[100, 10,Any,Any]	Edge-only	[14,241,0]	Edge-only	Edge-only	Edge-only	Edge-only
[1000, 0.1,Any,Any]	NE-only	[12,241,0]	NE-only	NE-only	NE-only	NE-only
[1000, 1,Any,Any]	[35,156,0]	[12,241,0]	NE-only	NE-only	NE-only	NE-only
[1000, 10,Any,Any]	Edge-only	[14,241,0]	Edge-only	Edge-only	[82,345,0]	[35,313,0]

TABLE 9. VGG16 model prefers through single tier execution for minimal costs.

Partition Type	Conditions
Edge-only	[≤ 1,Any, Any,Any], [10, ≥ 1, Any,Any], [100, 10,Any,Any]
NE-only	[10, 0.1, ≤ 100, Any], [10, 0.1, 1000, ≥ 1], ≥ 100, ≤ 1, ≤ 100 ,Any], [1000, 10, ≤ 100,Any], [1000, 10, 1000, ≥ 1]
Cloud-only	[10, 0.1, 1000, 0.1], ≥ 100, ≤ 1, 1000, 0.1], [1000, 10, 1000, 0.1]

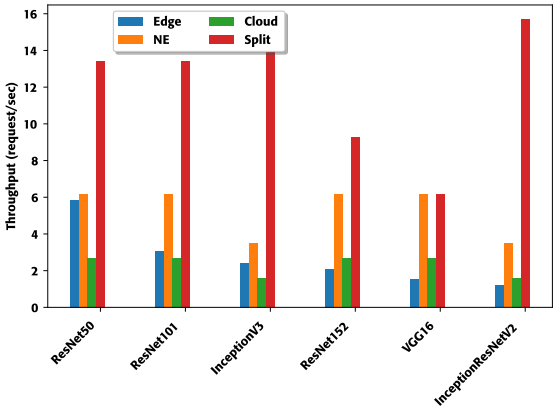


FIGURE 6. Throughput improvement achieved by *DNNSplit* for different models as compared to the single-tier execution.

the scenario in Figure 4, where the bandwidth between the Edge and NE is less, and even with increased bandwidth in Figure 4b the Jetson demonstrates the capability to execute entire models efficiently, especially advantageous for smaller models.

- **Cost Ratio Impact:** The algorithm adapts its partitioning strategy based on the cost ratio between the NE and Cloud instances, demonstrating responsiveness to the infrastructure and network cost disparities. In the scenarios, characterized by the lower transmission costs from NE-C, *DNNSplit* allocates more computation to the Cloud for larger models, leveraging cost advantage.
- **Bandwidth influence:** The algorithm responds to changes in bandwidth availability. In the scenario with higher bandwidth between E-NE and NE-C, the algorithm allocates more computation to the NE and Cloud, emphasizing the cost reduction associated with increased bandwidth. Also, we observe that, with increased bandwidth, there is a tendency for the algorithm to shift split positions to earlier layers, thereby directing more computation to the NE and Cloud tiers. This demonstrates how a decrease in computation costs can effectively counterbalance the rising transmission costs, given that the earlier layers correspond to higher data transmission according to Eq 15.
- **Model-Specific Partitioning:** Various DNNs demonstrate distinct partitioning approaches influenced by the balance between remaining computational demands and transmission requirements at potential split points. For instance, as shown in Table 9, a sequential model like VGG16 tends not to favor splitting, given its comparable transmission requirements across different split positions. Also, smaller models as shown in Table 8 only prefer two-tier partitioning even in case of high bandwidths and low costs between NE-C. However, as the model size increases, the splitting position adjusts, correlating with a reduction in cost to even result in a three-tier partitioning as shown in Table 10.

TABLE 10. Models that achieve minimal costs through up to three-tier partitioning.

Condition	ResNet152	InceptionResNetV2
[1, 0.1,Any,Any]	Edge-only	[42,782,0]
[1, > 0.1, Any,Any]	Edge-only	Edge-only
[10,≤ 1, *Any*, *Any*]	[40,517,0]	[42,782,0]
[10, 10,Any,Any]	[484,517,0]	[619,782,0]
[10, 0.1, 100, 0.1]	[122,517,0]	[23,42,782]
[10, 0.1, 100, ≥ 1]	[122,517,0]	[42,782,0]
[10, 0.1, 1000, 0.1]	[122,0,517]	[19,23,782]
[10, 0.1, 1000, ≥ 1]	[122,517,0]	[42,782,0]
[100, 0.1, (10, 100) ,Any]	NE-only	NE-only
[≥ 100, ≤ 1, 1000, 0.1]	Cloud-only	[0,23,782]
[≥ 100, 0.1, 1000, ≥ 1]	NE-only	NE-only
[100, 1, ≤ 10, Any]	[122,517,0]	[42,782,0]
[100, 1, 100, 0.1]	[122,517,0]	[23,42,782]
[100, 1, ≥ 100, ≥ 1]	[122,517,0]	[42,782,0]
[100, 10,Any,Any]	Edge-only	Edge-only
[1000, ≤ 1, ≤ 100, Any]	NE-only	NE-only
[1000, 1, 1000, ≥ 1]	NE-only	NE-only
[1000, 10, 100, 0.1]	[122,517,0]	[23,42,782]
[1000, 10, ≥ 100, ≥ 1]	[122,517,0]	[42,782,0]
[1000, 10, 1000, 0.1]	[122,0,517]	[19,23,782]

F. COMPARISON WITH PREVIOUS WORK

In this subsection, we compare the *DNNSplit* algorithm with previous works discussed in Sec. II.

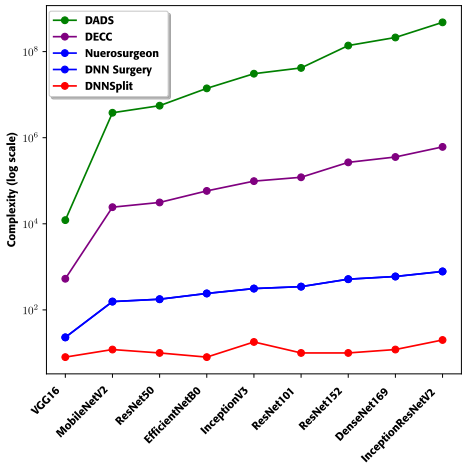


FIGURE 7. Complexity comparison of *DNNSplit* with lit. for nine DNNs.

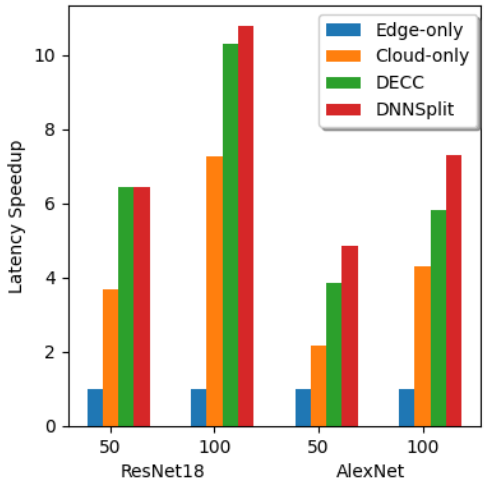


FIGURE 8. Latency speedup achieved by *DNNSplit* for the models used in the literature.

TABLE 11. Latency speedup comparison of *DNNSplit* with literature in a two-tier environment.

2*Approach	ResNet18		AlexNet	
	4G	WiFi	4G	WiFi
DNNSplit	6.1	18.7	7.6	12
DADS & DNN Surgery	6.45	8.08	7.23	8.32

1) COMPLEXITY COMPARISON

The effectiveness of an adaptive strategy depends significantly on both the search space and the algorithmic complexity required to adjust decisions in response to environmental changes. This is particularly crucial when dealing with DNN models, known for their extensive layer count. In this section, we evaluate the complexity of the *DNNSplit* strategy across nine models, comparing it to earlier approaches discussed in Section II. Figure 7 shows *DNNSplit's* superior complexity performance relative to other methods. Notably, the algorithmic complexity of *DNNSplit* remains relatively stable, even as DNN model sizes increase. In contrast, the other solution approaches in the figure exhibit a noticeable increase in complexity with larger model sizes, making some of these methods impractical for DNNs with many layers.

We note that our goal in this paper is to introduce *DNNSplit* and its superior algorithmic complexity. We do not claim overall superiority in latency, compute resource cost, or throughput compared to all existing works. Instead, our key contribution is in efficiently finding the lowest-latency split-point solution, making our method applicable to larger models where previous approaches faced challenges. This result demonstrates the effectiveness and scalability of the *DNNSplit* in optimizing adaptive decisions for DNNs of varying sizes.

2) LATENCY AND COST COMPARISON

We consider the two-tier configuration from [10] and [12], where a Raspberry Pi 3 model B serves as the Edge device and an Ali cluster machine with a 2.5 GHz and 8-core processor acts as the Cloud. We use the same 4G and WiFi bandwidths, neglecting the propagation delay between Edge and Cloud. None of the previous works used large models as used in this work for their evaluation most likely due to scalability issues. Hence, for comparison, we assess the performance of the *DNNSplit* algorithm with the two small models explored in the cited papers, namely ResNet18 and AlexNet. Figure 8 illustrates the latency speedup achieved by *DNNSplit* compared to the DECC strategy across two different bandwidth conditions 50 and 100 Mbps.

On the x-axis, we show two different bandwidths for each ResNet18 and AlexNet and the y-axis shows the speedup value. The figure shows that for the ResNet18 model, both DECC and *DNNSplit* exhibit comparable speedups. However, for the AlexNet model, *DNNSplit* achieves a slightly higher speedup than DECC. This shows that *DNNSplit* leverages the increased bandwidth to offload more layers to the Cloud. Table 11 compares the latency speedups of the various algorithms in a 2-tier environment. Notably, in contrast to *DNNSplit*, the latency speedup observed in DNN Surgery with a bandwidth increase beyond 18.88 Mbps (i.e. WIFI) is not substantial.

As discussed in Sec.II, there is currently no literature that thoroughly examines the cost associated with deploying DNN applications in a multi-tier environment. Consequently, the cost benefits of *DNNSplit* cannot be compared to any existing work in the literature. However, when we specifically assess the Edge-only cost comparison, similar to the approach taken for latency, we observe an improvement of up to 6 times, as illustrated in Figure 5, while still meeting stringent QoS requirements.

VIII. CONCLUSION AND FUTURE WORK

This paper presents the *DNNSplit* algorithm, a novel approach designed to identify potential split points for the efficient placement of DNN applications within a multi-tier architecture while considering QoS and SLA constraints. Our unique methodology involves detecting split positions in diverse DNNs that result in reduced latency and cost. These split

points, determined by our strategy, remain effective regardless of infrastructure configuration and network conditions. We demonstrate that our identified split points are the sole candidates for improving latency and cost; no other split point can achieve lower latency or cost. This significant contribution addresses the scalability issues of previous methods. Additionally, we develop a two-stage algorithm capable of finding split points that minimize latency and cost, adjusting to network fluctuations and varied infrastructure conditions. Importantly, the linear complexity of our algorithm enables its versatile application across DNNs of varying sizes within a multi-tier computational environment. *DNNSplit* achieves up to a 100-fold complexity improvement for larger models compared to existing strategies. We aim to improve our algorithms by integrating energy-efficient strategies, acknowledging the importance of reducing carbon footprint. Our future efforts include extending these algorithms to larger models like GPT and transformers, broadening our strategic impact.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[2] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surveys*, vol. 52, no. 6, pp. 1–36, Nov. 2020.

[3] D. Tennenhouse, "Surprise-inspired networking," *Computer*, vol. 56, no. 1, pp. 30–41, Jan. 2023.

[4] C. Jiang and J. Wan, "A thing-edge-cloud collaborative computing decision-making method for personalized customization production," *IEEE Access*, vol. 9, pp. 10962–10973, 2021.

[5] D. Xu, X. He, T. Su, and Z. Wang, "A survey on deep neural network partition over cloud, edge and end devices," 2023, *arXiv:2304.10020*.

[6] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.

[7] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, May 2017.

[8] J. Chen, Q. Qi, J. Wang, H. Sun, and J. Liao, "Accelerating DNN inference by edge-cloud collaboration," in *Proc. IEEE Int. Perform., Comput., Commun. Conf. (IPCCC)*, Oct. 2021, pp. 1–7.

[9] D. Luger, A. Aral, and I. Brandic, "Cost-aware neural network splitting and dynamic rescheduling for edge intelligence," in *Proc. 6th Int. Workshop Edge Syst., Anal. Netw.*, May 2023, pp. 42–47.

[10] H. Liang, Q. Sang, C. Hu, D. Cheng, X. Zhou, D. Wang, W. Bao, and Y. Wang, "DNN surgery: Accelerating DNN inference on the edge through layer partitioning," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 3111–3125, Jul. 2023.

[11] G. Liu, F. Dai, X. Xu, X. Fu, W. Dou, N. Kumar, and M. Bilal, "An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing," *Future Gener. Comput. Syst.*, vol. 140, pp. 422–435, Mar. 2023.

[12] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2019, pp. 1423–1431.

[13] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge, and end devices," *IEEE Trans. Ind. Informat.*, vol. 16, no. 8, pp. 5456–5466, Aug. 2020.

[14] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 854–863.

[15] C. Li, L. Chai, K. Jiang, Y. Zhang, J. Liu, and S. Wan, "DNN partition and offloading strategy with improved particle swarm genetic algorithm in VEC," *IEEE Trans. Intell. Vehicles*, Dec 2023, doi: 10.1109/TIV.2023.3346506.

[16] M. Xue, H. Wu, G. Peng, and K. Wolter, "DDPQN: An efficient DNN offloading strategy in local-edge-cloud collaborative environments," *IEEE Trans. Services Comput.*, vol. 15, no. 2, pp. 640–655, Mar. 2022.

[17] L. Yang, X. Shen, C. Zhong, and Y. Liao, "On-demand inference acceleration for directed acyclic graph neural networks over edge-cloud collaboration," *J. Parallel Distrib. Comput.*, vol. 171, pp. 79–87, Jan. 2023.

[18] H. Qi, F. Ren, L. Wang, P. Jiang, S. Wan, and X. Deng, "Multi-compression scale DNN inference acceleration based on cloud-edge-end collaboration," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 1, pp. 1–25, Jan. 2024.

[19] *AWS GPU Instance*. Accessed: Dec. 10, 2023. [Online]. Available: https://aws.amazon.com/ec2/instance-types/p4/

[20] *Raspberry Pi 4*. Accessed: Dec. 10, 2023. [Online]. Available: https://vilros.com/products/raspberry-pi-4-model-b-8gb-ram?src=raspberrypi

[21] *Jetson Nano*. Accessed: Dec. 10, 2023. [Online]. Available: https://www.arrow.com/en/products/900-13448-0020-000/nvidia

[22] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," Dept. EECS, Univ. California, Los Angeles, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.

[23] *Benchmark Models*. Accessed: Dec. 10, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications

PARIDHIKA KAYAL (Graduate Student Member, IEEE) received the bachelor's degree in technology (computer science) from the International Institute of Information Technology, Hyderabad, Telangana, India, in 2008, and the master's degree in computer science from North Carolina State University, Raleigh, NC, USA, in 2017. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada.

ALBERTO LEON-GARCIA (Life Fellow, IEEE) is currently a Professor of electrical and computer engineering with the University of Toronto. He authored the textbooks: Probability and Random Processes for Electrical Engineering, and Communication Networks: Fundamental Concepts and Key Architecture. He was the Founder and a CTO of AcceLight Networks in Ottawa from 1999 to 2002. AcceLight developed a multi-service switch-router with all-optical fabric. He was the Scientific Director of the NSERC Strategic Network for Smart Applications on Virtual Infrastructures (SAVI), and the Principal Investigator of the project on Connected Vehicles and Smart Transportation. SAVI designed and deployed a national testbed in Canada that converges cloud computing and software-defined networking. CVST designed and deployed an application platform for smart transportation. His research is currently focused on the design of intelligent automated management systems for large-scale infrastructures. He is a fellow of the Royal Society of Canada, and a Life Fellow of the Institute of Electronics and Electrical Engineering "For contributions to multiplexing and switching of integrated services traffic." He was the Founder of StreamWorx.ai which developed massive-scale, real-time streaming analytics software for network operations and cybersecurity applications. StreamWorx was acquired and is now part of OpenText. He has held several the Chair Positions at the University of Toronto: Nortel Chair in Network Architecture and Services; Skoll Chair in Computer Networks and Innovation; Canada Research Chair in Autonomic Service Architecture; and Distinguished Professor in Smart Infrastructures.