DISNET: Distributed Micro-Split Deep Learning in Heterogeneous Dynamic IoT

Eric Samikwa, Antonio Di Maio, and Torsten Braun, *Member, IEEE*

***Abstract*—The key impediments to deploying deep neural networks (DNNs) in Internet of Things (IoT) edge environments lie in the gap between the expensive DNN computation and the limited computing capability of IoT devices. Current state-of-the-art machine learning models have significant demands on memory, computation, and energy and raise challenges for integrating them with the decentralized operation of heterogeneous and resource-constrained IoT devices. Recent studies have proposed the cooperative execution of DNN models in IoT devices to enhance the reliability, privacy, and efficiency of intelligent IoT systems but disregarded flexible fine-grained model partitioning schemes for optimal distribution of DNN execution tasks in dynamic IoT networks. In this article, we propose distributed micro-split deep learning in heterogeneous dynamic IoT (DISNET). DISNET accelerates inference time and minimizes energy consumption by combining vertical (layer based) and horizontal DNN partitioning to enable flexible, distributed, and parallel execution of neural network models on heterogeneous IoT devices. DISNET considers the IoT devices' computing and communication resources and the network conditions for resource-aware cooperative DNN Inference. Experimental evaluation in dynamic IoT networks shows that DISNET reduces the DNN inference latency and energy consumption by up to 5.2× and 6×, respectively, compared to two state-of-the-art schemes without loss of accuracy.**

***Index Terms*—Distributed machine learning (ML), edge computing, Internet of Things (IoT), micro-split deep learning (DL).**

I. Introduction

INTERNET of Things (IoT) systems generate large volumes of data from user devices. The raw data generated by IoT devices is very often private or sensitive and can be too large to transmit over the networks [1]. For example, wearable devices, such as Google Glass or Apple Watch gather sensitive data by recording the daily activities of users. Similarly, the data collected from medical sensors [2], microphones [3], and cameras [4] are very often in large quantities and highly sensitive. This data is essential for executing machine learning (ML) models in order to deliver personalized services and other intelligent IoT applications [5], [6]. Consequently, there is a growing demand for implementing ML methods without directly accessing and aggregating sensitive raw data from devices at central servers.

Manuscript received 9 June 2023; accepted 27 August 2023. Date of publication 8 September 2023; date of current version 6 February 2024. *(Corresponding author: Eric Samikwa.)*

The authors are with the Institute of Computer Science, University of Bern, 3012 Bern, Switzerland (e-mail: eric.samikwa@unibe.ch; antonio.dimaio@unibe.ch; torsten.braun@unibe.ch).

Digital Object Identifier 10.1109/JIOT.2023.3313514

Deep neural networks (DNN) have shown extraordinary power in understanding large-scale data that are massively diverse and complex in several applications [7]. Because of their proximity to the data, conventional consumer-level devices, such as IoT devices, are great candidates for the in-the-edge processing of DNNs [8]. However, current state-of-the-art ML models have significant demands on memory, computation, and energy [9], [10]. This is incompatible with the resource-constrained nature of IoT devices that are characterized by limited energy budget, memory, and computation capability [11]. Model optimization methods, such as weight pruning [12] and precision reduction [13], [14], enable the running of limited versions of models on IoT devices. However, with the continuous advancement of DNNs models for various fields, these approaches are impractical for a wide range IoT applications, such as health, industrial, and multimedia-based [15], [16] because a restricted model (i.e., whose computational demands are adequately reduced to fit on IoT devices) may provide insufficient accuracy performance for such sensitive tasks. It is, therefore, necessary to devise methods to deploy models across a network of resource-constrained devices.

Distributed ML methods allow joint execution of DNN models by sharing the computation on multiple devices [16]. Such techniques have great potential to benefit from rich data generated by the distributed heterogeneous IoT devices without transmitting vast amounts of raw data over the central networks [8]. Among others, distributed ML in edge IoT environments bring the following opportunities and challenges: reducing the dependence on cloud resources and high-performance network infrastructure for scenarios with limited Internet connectivity, protecting private or sensitive user data by not exposing it outside the local network, and providing an alternative solution for understanding raw data locally other than the current standard solution of offloading to the cloud.

Deep learning (DL) driven applications are computation-intensive because of the depth of the DNN models (i.e., their massive number of layers) and large input dimension [9], [17], [18]. Most existing works apply *vertical* partitioning (Fig. 1) where the model is partitioned layer by layer, and some of the layer computations are processed in devices and some at the server or other devices [19], [20], [21]. The layer-based partitioning can solve the depth problem of DL-driven applications and reduce the computation burden on end devices. However, layer-based partitioning lacks a solution for the big input data and disregards the advantages of

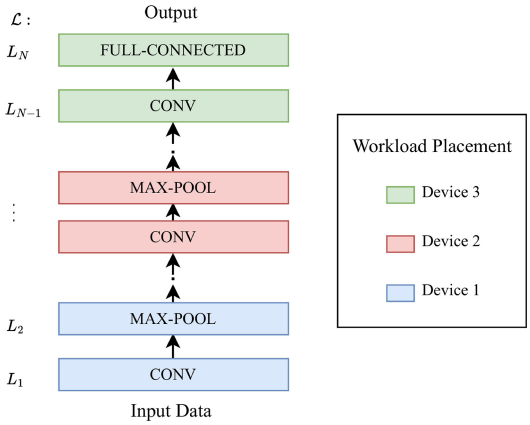

Fig. 1. Layer-based (vertical) DNN model partitioning on three devices.

intralayer parallelism since the layers are executed in a specific sequential order. Furthermore, the approach misses the opportunity to effectively exploit the collective computing power of numerous heterogeneous, ubiquitous, and decentralized IoT devices at the edge. In most variations of this approach, the last model layers are processed on the server, creating server dependency.

Cooperative execution of DNN models among edge devices has significant potential in reducing latency and energy costs by harvesting underutilized computing resources at the edge. Other recent studies have proposed data partitioning with horizontal DNN partitioning [22], [23], [24]. These allow parallel execution of model layers on multiple devices and distribute model input data (Fig. 2). However, the approaches pay less attention to the heterogeneity and variability of computing and network resources of IoT networks. Furthermore, only input-wise partitioning can increase data dependence, leading to overlapped computation and redundant communication.

To effectively exploit computing resources in edge IoT environments, we must consider the resource-constrained nature and limited energy budget of IoT devices and their heterogeneity. Furthermore, given the highly dynamic IoT network conditions, an optimal workload allocation that jointly considers computation and communications costs to improve the latency and energy consumption is desired. More specifically, it is crucial to decide which devices to participate in the cooperative ML, how much workload to allocate to each device and the communication efficiency among the devices. Besides, the topology of IoT mesh networks can be utilized to optimize communication by allocating the computation of partitions to devices based on the communication paths and quality of the communication channels among them.

In this article, we propose distributed micro-split deep learning in heterogeneous dynamic IoT (DISNET), a scheme for distributed DNN computing in resource-constrained heterogenous IoT devices for low-latency and energy-efficient cooperative DL. DISNET models the neural network as a weighted directed acyclic graph (DAG) that can be flexibly partitioned into a network of IoT devices. By considering both vertical and horizontal DNN partitioning, DISNET enables efficient and parallel processing of a DNN model while utilizing the collective computing of heterogeneous, ubiquitous, and decentralized IoT devices. The IoT network is considered as a weighted graph where the weights correspond to the available computation and network resources on the devices and their respective wireless channels. DISNET dynamically splits and allocates the processing of neural network partitions to the IoT devices in the network for optimizing DNN inference tasks. We envision the deployment of DISNET in environments, such as smart home, Industrial IoT, Internet of Medical Things (IoMT), smart environmental monitoring, etc., wherein the devices are willing to cooperate and share their resources. In summary, this article makes the following contributions.

1) We propose *micro-split deep learning* to enable flexible partitioning and distributed computing of neural network models in heterogeneous dynamic IoT for low-latency and energy-efficient cooperative DNN inference.
2) We model the problem of flexible fine-grained neural network model splitting and device allocation, comprising arbitrary-sized DNN partitions, and dynamic mesh networks with heterogeneous resources. We prove the NP-hardness of the problem and formulate its relaxation.
3) We propose an efficient heuristic based on the relaxed problem that iteratively combines vertical and horizontal DNN partitioning for distributed execution of neural networks in dynamic IoT with diverse computing capabilities and network conditions without compromising accuracy.
4) We consider the tradeoffs between minimizing the inference execution time and minimizing the energy consumption on the IoT devices by introducing an energy-sensitivity parameter for the optimization that accounts for the application's energy or time constraints (requirements) in cases of tradeoffs between the two objectives.
5) We implement a multidevice prototype comprising of heterogeneous IoT devices and evaluate DISNET against other state-of-the-art approaches in various dynamic scenarios to corroborate its superior performance.

II. Related Work

Neurosurgeon [30] is a benchmarking-based approach in which a regression-based method is used for estimating optimal partitions of DNN models between a device and the cloud during installation. Scission [25] is a tool for automated benchmarking of DNN on a given set of the target device, edge, and cloud resources for determining optimal partitions that maximize performance. The decision making is based on the hardware capabilities of the target resources and their locality. However, partitioning the model between edge devices and the cloud is limited by the long communication distance between the cloud and the IoT devices, which may cause excessive communication latency, therefore, fail to meet real-time response requirements in time-critical applications. Furthermore, these approaches increase the risk of privacy loss as the data is shared outside the edge network.

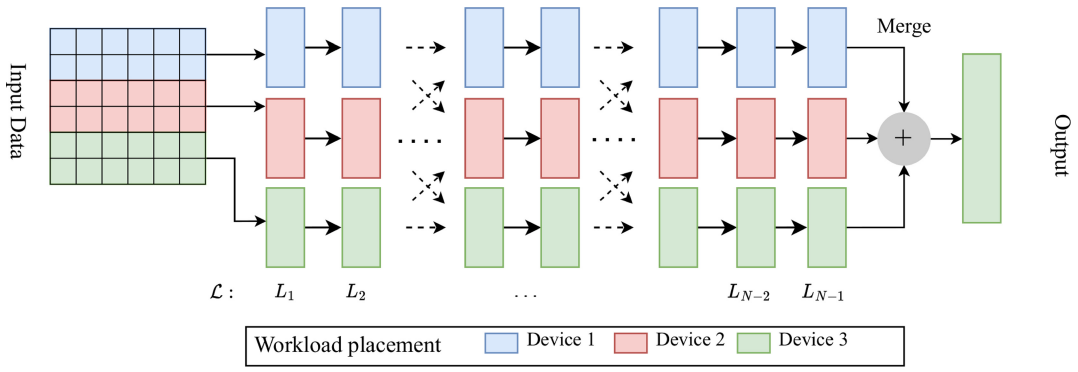

Fig. 2. Cooperative model execution on three devices through horizontal DNN partitioning to allow parallel execution of the model layers.

Alternatively, edge computing is a promising approach that allows IoT devices to offload computation-intensive tasks to the edge servers in their proximity to reduce inference time [21], [31], [32]. Edgent [26] employs DNN partitioning to utilize the benefits of collaborative edge intelligence between the edge and mobile devices. BranchyNet [33], which provides an architecture for the early exit mechanism, allows the inference results for the testing samples to exit the computation early when the result is already in high confidence by using entropy. Boomerang [4] uses DNN right-sizing and partitioning for on-demand cooperative inference execution between the IoT device and the edge server. However, these approaches only focus on the partitioning of sequential models in one dimension (i.e., layer-based partitioning of models between a device and edge server), which may not always result in an optimal distribution of the model execution task. Furthermore, these approaches do not fully utilize the collective edge computing and communication capabilities of the IoT devices.

To distribute DNN processing tasks among edge devices, model, and data partitioning methods are utilized. Model partitioning splits the model and distributes submodels to different devices. SPINN [34] splits convolutional neural networks (CNN) models into device and server submodels for distributed inference in resource-constrained environments. However, vertical splitting of a model between a device and server may not always result in optimal performance for heavily resource-constrained devices and misses utilizing the collective edge computing power of the IoT devices. Krouka et al. [27] proposed model compression and splitting for energy-efficient collaborative inference over time-varying channels. However, their approach affects the model accuracy for the inference.

Data partitioning distributes data partitions to devices and executes them in parallel. MoDNN [23] proposed input data partitioning scheme for convolutional and fully connected layers. CoEdge [22] proposes input image piece-wise partitioning for cooperative DNN inference over heterogeneous edge devices. DeepThings [24] utilizes the same technique to distribute inference tasks on end devices, such as Raspberry Pi and Android smartphones. The partitioning choice is based on the computation capabilities and memory of the end devices. However, input-wise partitioning only can result in increased data dependence, leading to overlapped computation and redundant communication and synchronization tasks.

Hadidi et al. [29] studied the optimal partitioning method for each model layer through input and output splitting for single-batch inferences of DNNs [29]. However, their approach does not consider the heterogeneity and variability of computing and network resources of IoT networks and the synchronization cost between devices. DeepSlicing [28] empirically partitions a CNN model via data partitioning and model partitioning, leading to a flexible partitioning of the DNN execution. However, DeepSlicing substantially increases the subinputs on each collaborating device, causing excessive computation, and communication overhead. Furthermore, DeepSlicing causes a loss of accuracy for the inference.

EosDNN [35] proposed an offloading scheme for DNN inference acceleration in a local–edge–cloud collaborative environment. EosDNN focuses on large-scale DNN parallelism in the multiuser local–edge–cloud collaborative environment. The approach in DDPQN [36] improves DNN task allocation with low energy consumption, low delay, and low cost in a local–edge–cloud collaborative environment to improve the quality of service. Their approach focuses on the model layer distribution in the local–edge–cloud collaborative environment with node balance. However, both solutions do not address the flexible partitioning and allocation of DNN partitions among resource-constrained heterogeneous IoT devices for cooperative execution at the edge.

Other studies have focused on model reduction techniques to minimize the computation requirements of computationally intensive DNN models, such as weight pruning [37], resource partitioning [38], quantization and low-precision inference [39], and binarizing weights [40]. However, these approaches require several additional steps that decrease the accuracy and enforce retraining of the model. DISNET is orthogonal to them and can run with these specified technologies after necessary adaptions. Moreover, our work pays more attention to resource-constrained and decentralized heterogeneous IoT devices.

In our previous work, we introduced *early exit of computation (EEoC)* for Energy-Efficient and Low-Latency Machine Learning over IoT Networks [41]. EEoC is an adaptive approach for distributing computation of ML inference over IoT networks for latency and energy optimization. EEoC

TABLE I
SUMMARY OF THE STATE-OF-THE-ART DISTRIBUTED ML TECHNIQUES IN IOT, COMPARED WITH DISNET'S FEATURES: A = D2D COLLABORATION, B= E2E LATENCY, C=ENERGY CONSUMPTION, D = VARIABLE RESOURCES, E = FINE-GRAINED MODEL PARTITIONING, AND F= IOT MESH NETWORKS

Work	A	B	C	D	E	F
Scission [25]		✓				
Edgent [26]		✓		✓		
Krouka et. al. [27]		✓	✓			
MoDNN [23]	✓	✓				
CoEdge [22]	✓	✓	✓			
DeepThings [24]	✓			✓		
DeepSlicing [28]	✓	✓			✓	
Hadidi et al. [29]	✓	✓				
DISNET	✓	✓	✓	✓	✓	✓

considers the time-varying network conditions and available computing resources to determine the optimal distribution of DNN models between the IoT device and edge by selecting an ideal model split point. EEoC improves inference efficiency for ML tasks in IoT systems but does not consider distributing the model among multiple devices for cooperative inference or training utilizing both model and data partitioning.

DISNET introduces micro-split neural network partitioning for low-latency and energy-efficient cooperative DL over a network of IoT devices. DISNET models the neural network as weighted DAG that can be partitioned in various degrees of freedom to minimize the inference time and energy consumption. The IoT network is considered a weighted directed graph, where the weights correspond to the available computation and network resources on the devices and their respective wireless channels. DISNET dynamically aligns the neural network partitions to the IoT network for optimal execution of the inference tasks without loss of accuracy.

Unlike classic DNN partitioning approaches, DISNET enhances privacy preservation within heterogeneous dynamic IoT environments through two key mechanisms. First, the data remains confined within the local network of devices and is potentially not transmitted to external servers for processing, which is common in vertical DNN partitioning approaches. As a result, the sensitive data remains within the controlled environment of the IoT network, reducing the risk of unauthorized access or data breaches during transmission. Second, by dividing the neural network model at both the layer and horizontal dimensions, different data portions are distributed across multiple devices, making it significantly harder for an unauthorized party to reconstruct the complete data or gain comprehensive insights into the original input data. Thus, DISNET enables low-latency and energy-efficient cooperative DL in dynamic heterogenous IoT while maintaining a high level of privacy. Table I shows a summary of the limitations of state-of-the-art distributed ML techniques in IoT, compared with DISNET's features. The comparison is based on device-to-device (D2D) collaboration (A), end-to-end (E2E) latency (B), energy consumption (C), variable computing and network resources (D), flexible fine-grained partitioning (E), and consideration for IoT mesh networks (F).

TABLE II
NOTATIONS TABLE

Notation	Description
Φ	Set of devices in the IoT mesh network
$\mathcal{L}$	A set of layers of a DNN model
G_ϕ	Graph of connected IoT devices in a mesh network
$A^{(k)}$	Adjacency matrix for the IoT mesh network graph
ξ	Set of edges in the IoT mesh network graph
$\rho(\phi)$	Computing rate of a given model partition for device ϕ
$B_{i,j}^{(k)}$	Network throughput for wireless data transfer from device i to device j at round k
G_λ	Neural network DAG representation of a model
$H^{(k)}$	Adjacency matrix for the neural network DAG
ϵ	Set of edges in the neural network DAG
$V_{i,j}$	Inter-partition data volume from partition i to partition j
ψ	Kernel size for a convolutional layer
s	Stride for a convolutional kernel
c_{in}	Number of input channels for a model layer
c_{out}	Number of output channels for a model layer
$\delta_{c,\lambda}^{(k)}(\phi)$	Computation time on a device ϕ required to execute model partition λ in time slot k
$E_{c,i}^{(k)}(\phi)$	The energy consumption of device ϕ required to execute model partition λ_i in time slot k
$\delta_{x,i}^{(k)}(\phi)$	Time required to exchange the intermediate data for partition λ_i
$E_{x,i}^{(k)}(\phi)$	The energy consumption of devices required to exchange intermediate data for processing partition λ_i in time slot k
$\Delta^{(k)}$	Time taken by the system to perform a cooperative inference task
$E^{(k)}$	Total energy by the system to perform a cooperative inference task
$P_{\text{c}}(\phi)$	Device ϕ power consumption during computation
$P_{\text{t}}(\phi)$	Device ϕ power consumption during data transmission
$P_{\text{r}}(\phi)$	Device ϕ power consumption during receiving
λ_i	A partition of a neural network
γ_i	Height of the partition λ_i representing the number of rows that it covers on the input
ω_i	Width of the partition λ_i representing the number of layers that it covers
Λ	Set of model partitions representing nodes in the neural network DAG
Ω	Set of admissible sets of partitions that compose the DNN
a	Allocation vector indicating the devices that host partition(s)
α	Energy sensitivity coefficient

III. DISTRIBUTED MICRO-SPLIT DEEP LEARNING OVER COOPERATIVE HETEROGENEOUS DYNAMIC IOT

A. System Model

We consider a scenario containing a set of heterogeneous IoT devices denoted by the index set $\Phi = \{1, \ldots, |\Phi|\}$, where devices can communicate over their wireless interfaces to form a mesh network. In this scenario, the IoT devices cooperatively share their computing resources to compute inference of a DNN model. Table II shows the key notations used in the system model. We assume that the time in the system is divided into equal time slots. During the generic time slot k, an IoT device i can transmit data to an IoT device j via wireless link with an average throughput of $B_{i,j}^{(k)}$ [bit/s]. We model the network of connected IoT devices as the undirected graph $G_\Phi = (\Phi, \xi)$ with nodes Φ and edges ξ. The link weights between any two nodes $i, j \in \Phi$ during time slot k can be represented by a weighted adjacency matrix $A^{(k)} = \{A_{ij}^{(k)}\}$. The link weights in the adjacency matrix $i, j \in \Phi$ represent the

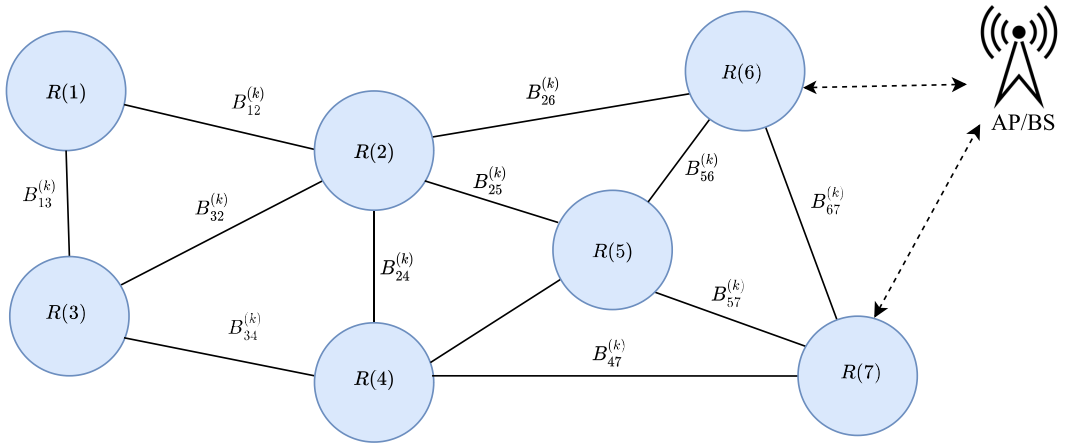

Fig. 3. Sample mesh network topology with seven IoT devices, edge weights B_{ij}^{k} represent the network throughput between devices i and j at time slot k, and each vertex is characterized by a resource vector $R(\phi)$ representing the device resources.

network throughput $B_{i,j}^{(k)}$ of transmitting data from device i to device j during time slot k. The adjacency matrix describes whether there is a communication link between any two nodes in the IoT mesh network and the capacity of the connections during time slot k

$$A_{i,j}^{(k)} = \begin{cases} B_{i,j}^{(k)}, & \text{if}(i,j) \in \xi \\ 0, & \text{if}(i,j) \notin \xi \end{cases} \quad \forall k > 0. \tag{1}$$

We assume that the system offers an inference service that requires running a DNN model of a fixed number of subsequent layers with similar or different characteristics (e.g., convolution layer and dense layer). Let us define the DNN model's computing intensity I [cycles/B] as the average number of computing cycles required to process 1-B input data. We denote device ϕ's CPU frequency $f(\phi)$ [cycles/s] as the average number of computing cycles per second. We can now define device ϕ's *computing rate* $\rho(\phi) = f(\phi)/I$ [B/s] as the volume of bytes input into the DNN processed by device ϕ per second [22]. Previous studies have shown that for a given model on a device, the computing rate can be estimated by application-driven benchmarking [22], [28], [42]. Let us define $P_c(\phi)$ and $P_t(\phi)$ as the average power, expressed in Watts, used by device ϕ for computation and wireless transmission, respectively. Several studies [43], [44], [45] carried out the profiling of power consumption of resource-constrained devices during the execution of DNN models. For the sake of simplicity, we assume that each device has a sufficient amount of available memory to allocate any portion of a DNN task. Each device $\phi \in \Phi$ is fully characterized by a *resource vector* $R(\phi) = (\rho, P_c, P_t)(\phi)$. Fig. 3 shows a sample mesh network graph with seven IoT devices at time slot k. Each edge weights B_{ij}^{k} represents the network throughput between devices i and j at time slot k. Each vertex is characterized by a resource vector $R(\phi)$, representing the resources owned by device ϕ.

Given a DNN model, $\mathcal{L} = \{1, \ldots, |\mathcal{L}|\}$ denotes the index set that identifies each DNN's layer in order. For a CNN model, each layer can be either a convolution, pooling, flatten, or dense layer. A *configuration vector* $(\psi, s, c_{\text{in}}, c_{\text{out}}, p, \Gamma)_l$ denotes the characteristics of layer $l \in \mathcal{L}$, namely, the convolution kernel size ψ_l, the stride s_l, the padding p_l, the number of input channels $c_{\text{in},l}$, the number of output channels $c_{\text{out},l}$, and the height of the input dimension Γ_l. We define the input feature map height Γ and width W as the number of rows and columns of the input data, respectively. It is worth noting that the configuration vector can flexibly represent convolution, pooling, and dense layers, as the latter can be modeled as a special case of the former. A pooling layer performs a nonlinear downsampling operation (e.g., max pooling or average pooling) over a small window and is described by the hyperparameters, such as pooling kernel size, stride, and padding. A dense layer is equivalent to a convolutional layer with input feature map's size $1 \times 1 \times c_{\text{in}}$, output feature map's size is $1 \times 1 \times c_{\text{out}}$, kernel size $\psi = 1$, stride $s = 1$, and padding $p = 0$ [22]. Similarly, for a flatten layer with input feature map's size $1 \times \Gamma \times c_{\text{in}}$, the output size is $1 \times \Gamma \times c_{\text{in}}$, with padding $p = 0$ and no kernel size or stride.

We define a *partition* λ_i as a part of a neural network that can be independently executed on a device. We define the *partition height* $\gamma_i \in \mathbb{N}$ as the *height* of the partition λ_i, which represents the number of rows of the input feature map covered by partition λ_i. We define the *partition width* $\omega_i \in \mathbb{N}$ as the *width* of the partition λ_i, which represents the number of layers covered by partition λ_i. It is worth highlighting that the partition width ω_i is not the width of the input feature map W. In this system model, any partition λ_i is completely defined by the couple (γ_i, ω_i), which gives the partition a "trapezoidal" shape. Let us define $\Lambda = \{\lambda_1, \ldots, \lambda_{|\Lambda|}\}$ as the set of partitions that composes the DNN, and Ω as the set in which each element is an admissible set $\Lambda \in \Omega$ of partitions that compose the DNN. For any given admissible set of partitions $\Lambda \in \Omega$, the union of all subsets with an individual partition is equivalent to the set of partitions, and so the DNN i.e., $\bigcup_{i=1}^{|\Lambda|} \{\lambda_i\} = \Lambda$. Similarly, the intersection of all subsets with an individual partition is equivalent to an empty set, i.e., $\bigcap_{i=1}^{|\Lambda|} \{\lambda_i\} = \emptyset$.

We represent a DNN model as a DAG $G_\Lambda = (\Lambda, \epsilon)$, where Λ is the set of vertices and ϵ is the set of edges. It is possible to represent a DNN as a DAG because the flow of data and computation sequence of DNN sublayers does not form loops on the DNN partitions, meaning that for a single inference, each partition is executed at most once. The elements H_{ij} of the

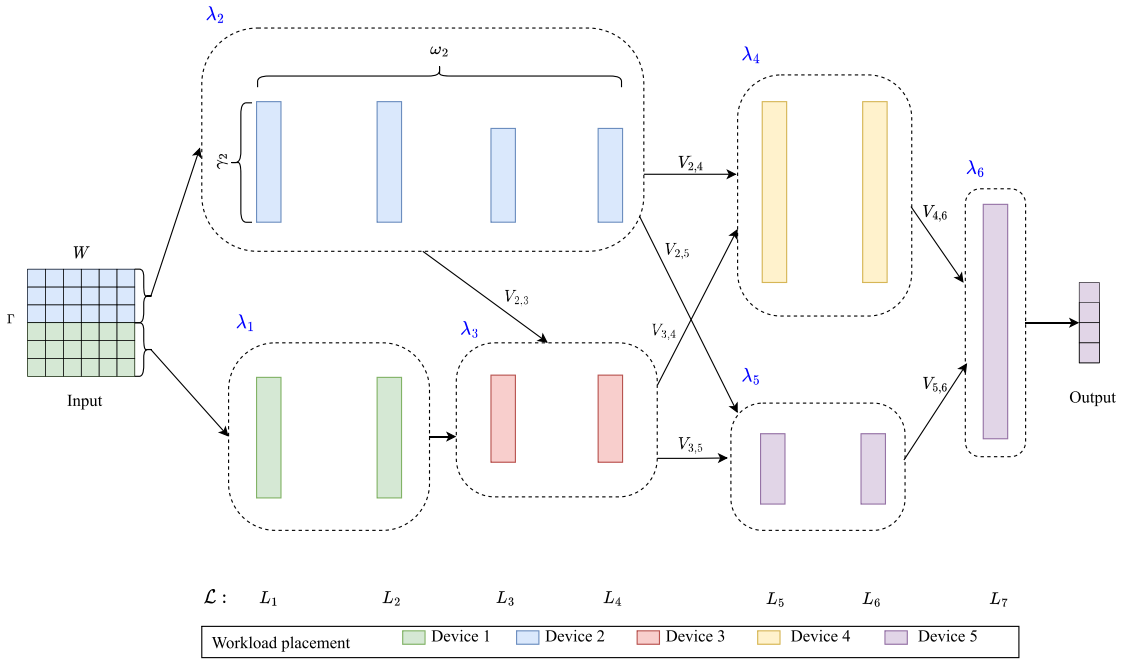

Fig. 4. Example of a neural network model with seven layers partitioned into $\Lambda = \{\lambda_1, \ldots, \lambda_6\}$, each partition λ_i has a height γ_i and width ω_i and the partitions are allocated to five devices, i.e., partition λ_5 and λ_6 allocated to Device 5.

weighted adjacency matrix H represent link weights between any two vertices $\lambda_i, \lambda_j \in \Lambda$, whose values represent the volume of the intermediate data $V_{i,j}$ produced by partition λ_i as output and fed to partition λ_j as input. In other words, the adjacency matrix describes whether there is a data exchange between any two partitions in the neural network DAG and the amount of data to be exchanged

$$H_{ij} = \begin{cases} V_{i,j}, & \text{if}(i,j) \in \epsilon \\ 0, & \text{if}(i,j) \notin \epsilon. \end{cases} \tag{2}$$

For any given split configuration G_Λ, the volume of the data transferred between partitions connected by an edge remains the same for all time slots k. Therefore, the adjacency matrix H does not depend on k.

Previous works consider only basic split configuration of a DNN, where each layer is a partition (i.e., a vertex in the neural network DAG represents a layer). Given a sequential DNN model with layers $\mathcal{L} = \{1, \ldots, |\mathcal{L}|\}$, the resulting partitions graph will have $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$ nodes connected sequentially. Differently from previous schemes, DISNET generalizes the concept of model splitting by removing the constraint on splits to include entire layers. In particular, a partition λ_i in DISNET covers layers partially over γ_i input rows, and each partition λ_i spans across ω_i consecutive layers. Any set of partitions Λ must comply with the directions in the graph G_λ described by H_{ij}. This is to ensure the correct order of execution for the model (or submodel) layers and the flow of intermediate data between them. Fig. 4 shows an example of a possible neural network partitioning computed by DISNET, where the DNN is partitioned in a set of partitions Λ, whose elements are connected by directed edges to form a DAG. Fig. 4 highlights that a device can host multiple partitions: this feature allows the flexible allocation of DNN portions of custom shapes as a set of "trapezoidal" shaped partitions.

We define the *interpartition data volume* $V_{i,j}$ as the data volume transmitted from partition λ_i to λ_j to perform one inference task. The value of $V_{i,j}$ depends on sizes (γ_i, ω_i) and (γ_j, ω_j), and the configuration vectors of the layers covered by the partitions λ_i and λ_j. The volume of data transmitted between devices depends on the interpartition data volume $V_{i,j}$, but also on the devices on which the partitions are allocated. We define the *allocation vector* $a = (a_1, \ldots, a_{|\Lambda|}) \in \Phi^{|\Lambda|}$ as a vector whose ith component is the index $a_i \in \Phi$ of the device that hosts partition $\lambda_i \in \Lambda$. Fine-grained partitioning significantly affects the communication between devices, such as for convolution operations that process data across partition boundaries. For example, if a layer is split between devices, the boundary data (i.e., a fixed symmetric number of rows on both sides of the split dimension) needs to be shared between the devices as padding data to allow complete convolution operations on each device [28]. When a layer l has kernel size $\psi_l > 1$, each device that holds a partition covering a portion of layer l needs to pull padding data of size $\lfloor \psi_l/2 \rfloor$ along the split dimension from the neighboring devices. For layers split into multiple partitions, each device that holds the partitions needs to exchange this data. This formulation can be used recursively to determine how much padding data will be sent in advance for partitions covering multiple layers.

After completing the computation needed to execute partition λ_i, the device hosting partition λ_i needs to send the intermediate data to its neighboring devices that hold any partitions connected to partition λ_i by an edge until the task is complete. Each layer's input features are the previous layer's output features, so it is possible to derive the exact size of the output the device needs to send after computing a

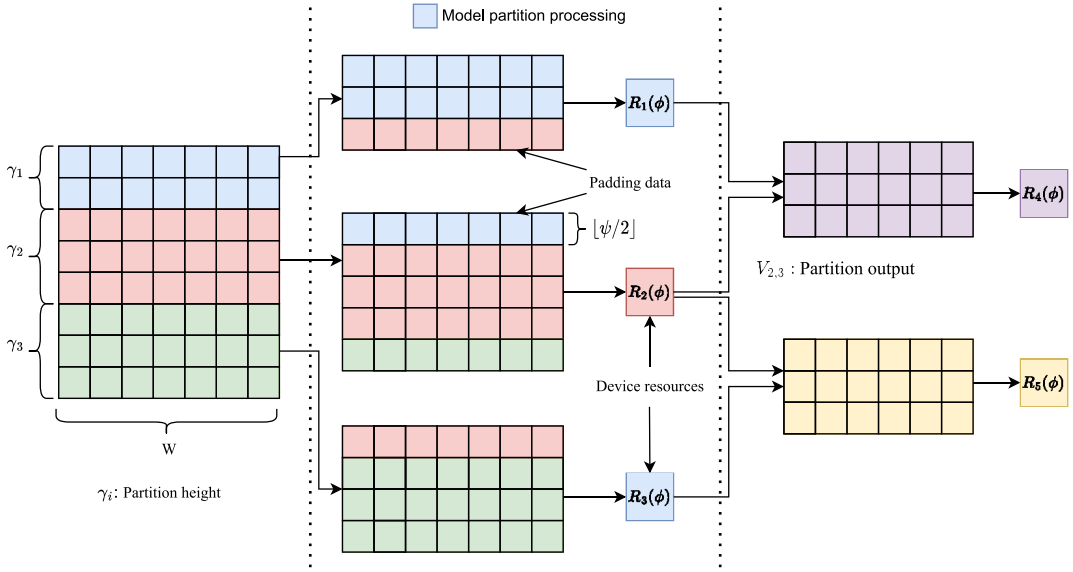

Fig. 5. Partitions data flow for an example scenario with five partitions on different devices with heterogeneous resources, each device processing a model partition λ_i and sharing the partition's input data and padding data of size $\lfloor \psi/2 \rfloor$.

partition. As a first example, assume a partition λ_i has a partition width $\omega_i = 1$ which is a portion of a convolutional layer l with configuration vector $(\psi, s, c_{\text{in}}, c_{\text{out}}, p, \Gamma)_l$. If an input feature map of size $(\gamma, W, c_{\text{in}})$ (height, width, channels) is fed to partition λ_i, the output size of partition λ_i is $([\gamma - \psi + 2p/s]+1, [W - \psi + 2p/s]+1, c_{\text{out}})$ (height, width, channels). Given that the configuration vectors of all the layers are known, this operation can be applied recursively to determine the output size of a partition covering more than one layer (i.e., $\omega_i > 1$) in which the output data from the partition is the same as the output of the portion of the last convolutional layer covered by the partition and its input is the output of the last but one layer covered by the partition. Similarly, if the last layer is a portion of a dense layer l with ϱ neurons and configuration vector $(\psi = 1, s = 1, c_{\text{in}} = 1, c_{\text{out}} = 1, p = 0, \varrho)_l$, the output of dense layers is determined by the number of its neurons. Therefore, if an input of size $(\gamma, 1, 1)$ (height, width, channels) is from the last but one layer covered by the partition, the output size of partition λ_i is $(\varrho, 1, 1)$ (height, width, channels). Fig. 5 shows an example scenario of the data flow for five partitions on different devices, including the partition input height and the padding data.

Partitioning the DNN for the IoT network introduces some numerical constraints. We assume that the available amount of memory on all devices Φ is enough for the DNN execution task and model storage. Let us define a helper function $\gamma_i(l)$ that returns the height of partition λ_i at layer l. For each layer l, the sum of the heights at l of all partitions covering partially l must be equal to the layer l's input size Γ_l

$$\sum_{i:\lambda_i \in \Lambda} \gamma_i(l) = \Gamma_l \quad \forall l \in \mathcal{L}. \tag{3}$$

Let Υ be the set of all possible paths between the start and the end partitions of the DAG G_λ. Let $\upsilon \in \Upsilon$ be a path in the DAG G_λ between the start and end partitions, where υ is a subgraph $\upsilon = (G_\upsilon, E_\upsilon)$ of G_λ. Notice that G_υ represents the set of the path's vertices, and E_υ represents the set of edges of the path. In any given path, the sum of the partition widths ω_i for all the partitions in the path $\lambda_i \in \upsilon$ is equal to the number of layers in the DNN model as shown in the following:

$$\sum_{i:\lambda_i \in \upsilon} \omega_i = |\mathcal{L}| \quad \forall \in \upsilon \in \Upsilon. \tag{4}$$

Finally, in order for a split to be admissible, any edge between any two partitions λ_i and λ_j allocated on different devices a_i and a_j must lie on an existing edge between a_i and a_j

$$\forall(\lambda_i, \lambda_j) \in \epsilon : a_i \neq a_j \implies (a_i, a_j) \in \xi. \tag{5}$$

If a partition λ_i has an input feature map of size $(\gamma, W, c_{\text{in}})$ we can define $r_i = \gamma \cdot W \cdot c_{\text{in}}$ as the data volume input to partition λ_i. Similarly, we define r_Θ as the data volume input to a reference partition λ_Θ used when determining the value of $\rho(\phi)$ in a benchmarking process. We can now define the *computation time* $\delta^{(k)}_{c,i}(\phi)$ required to execute partition $\lambda_i \in \Lambda$ on a device $\phi \in \Phi$ at time slot k

$$\delta^{(k)}_{c,i}(\phi) = \frac{r_i}{\rho(\phi) \cdot r_\Theta} \cdot \frac{\omega_i}{\omega_\Theta}. \tag{6}$$

The energy consumption of the device $\phi \in \Phi$ required to compute model partition $\lambda_i \in \Lambda$ at time slot k is the product of the computation time $\delta^{(k)}_{c,i}(\phi)$ and the device's power consumption $P_c(\phi)$ is determined in the following:

$$E^{(k)}_{c,i}(\phi) = \delta^{(k)}_{c,i}(\phi) \cdot P_c(\phi). \tag{7}$$

To compute the partition λ_i, input or padding data is required from other partitions. We define the *in-neighborhood* $N^-(\lambda_i) = \{\lambda_j \in \Lambda : (\lambda_j, \lambda_i) \in \epsilon\}$ as the set of partitions that send interpartition data to partition λ_i. We define the *out-neighborhood* $N^+(\lambda_i) = \{\lambda_q \in \Lambda : (\lambda_i, \lambda_q) \in \epsilon\}$ as the set of partitions that receive interpartition data from partition λ_i. The receiving time $\delta^{(k)}_{r,i}(\phi) = \sum_{\lambda_j \in N^-(\lambda_i)} (V_{j,i}/B^{(k)}_{j,i})$ is needed

to receive the interpartition data for the computation of a partition λ_i from the neighbors $N^-(\lambda_i)$. The transmission time $\delta_{t,i}^{(k)}(\phi) = \sum_{\lambda_q \in N^+(\lambda_i)} (V_{i,q}/B_{i,q}^{(k)})$ is needed to send the interpartition data after the computation of a partition λ_i to the neighbors $N^+(\lambda_i)$. The total time $\delta_{x,i}^{(k)}(\phi)$ required to exchange interpartition data for partition λ_i is shown in the following:

$$\delta_{x,i}^{(k)}(\phi) = \delta_{r,i}^{(k)}(\phi) + \delta_{t,i}^{(k)}(\phi). \tag{8}$$

For the sake of simplicity, (8) does not consider the queuing delays since we are optimizing inference for respective single input at a time. The energy consumption $E_{x,i}^{(k)}(\phi)$ of device $\phi \in \Phi$ required for the intermediate data exchange for processing partition λ_i in time slot k is the product of the wireless communication time and the devices power consumption given in the follows:

$$E_{x,i}^{(k)}(\phi) = \delta_{t,i}^{(k)}(\phi) \cdot P_t(\phi) + \delta_{r,i}^{(k)}(\phi) \cdot P_r(\phi). \tag{9}$$

The energy consumption for communication comprises transmission and receiving energy. For the initial partitions in the neural network, the input request may be generated within the device. As such, only the transmission energy is considered. Similarly, for the final partition, we consider the energy consumption for receiving intermediate data to complete the DNN execution task.

To determine the cost of the whole inference process, we must consider the processing delay. We denote $\Delta^{(k)}$ as the time the system takes to perform an inference task

$$\Delta^{(k)}(a) = \max_{\upsilon \in \Upsilon} \left\{ \sum_{i:\lambda_i \in G_\upsilon} \left(\delta_{c,i}^{(k)}(a) + \delta_{x,i}^{(k)}(a) \right) \right\}. \tag{10}$$

We acquire $\Delta^{(k)}$ by calculating the maximum path latency for the computation and transmission of all the devices within the path as shown in (10). This is because the flexible partition scheme applied to G_λ introduces parallelism in the cooperative execution of the DNN by allowing splitting of the computation load of the individual model layers. Thus, unlike layer-based partitioning of a DNN model where the layers are executed in a specified sequence, splitting layers among devices through the partitions allows parallel execution of the layers and the model.

We denote $E^{(k)}$ as the total energy consumption to perform the inference task. $E^{(k)}$ is determined as the sum of computation energy and communication energy required to process all the partitions of the neural network for devices involved in the task as shown in the following:

$$E^{(k)}(a) = \sum_{i:\lambda_i \in \Lambda} \left(E_{c,i}^{(k)}(a) + E_{x,i}^{(k)}(a) \right). \tag{11}$$

Minimizing the global inference time $\Delta^{(k)}$ and minimizing the IoT devices' total energy consumption $E^{(k)}$ are potentially conflicting objectives: for the IoT system, the partitions of the neural network graph Λ that minimizes energy consumption does not necessarily coincide with the partitions that minimize total inference time [41], [46]. Therefore, the actual value of the partition configuration of the neural network DAG that jointly optimizes inference time and energy depend on the considered application requirements. In particular, some application configurations may be energy sensitive, meaning that extending the devices' lifetime is more important than reducing the model computing time. Some other applications may be more time sensitive, meaning that minimizing time is more important than extending device's lifetime. We define the *energy sensitivity* $\alpha \in [0, 1]$ as a coefficient that represents the application's preferences between reducing inference time or energy consumption. We define α such that the closer α is to zero ($\alpha \to 0$), the lower the system's energy consumption. Conversely, the closer α is to 1 ($\alpha \to 1$), the shorter the global model's operation time. This work assumes that α does not change over time and is a parameter fixed by a policymaker that interprets the application's requirements.

This definition of α allows us to convert a multiobjective optimization problem that aims at minimizing inference time and energy consumption separately in a single-objective optimization problem. Each partition λ_i is defined by the height γ_i and width ω_i. The optimization problem aims to find a set of partitions $\Lambda = \{\lambda_1, \ldots, \lambda_N\}$ and an allocation vector a to a linear combination of total energy consumption and inference time. Hence, we can formulate the cooperative inference optimization as the following problem:

$$\begin{aligned} \underset{(\Lambda,a)\in\Omega\times\Phi^{|\Lambda|}}{\text{minimize}} \quad & \alpha \cdot \Delta^{(k)}(a) + (1-\alpha) \cdot E^{(k)}(a) \\ \text{subject to} \quad & (3),\ (4),\ (5). \end{aligned} \tag{12}$$

B. DISNET Architecture and Operation

The optimization problem $\mathcal{P}$ defined in (12) is an Integer Linear Programming (ILP) problem, which is NP-hard (Theorem 1). A typical IoT system, such as Industrial IoT, smart environmental monitoring systems, etc., may contain many connected devices. On the other hand, state-of-the-art DNN models have many layers and large input dimensions. Moreover, any feasible solution would need to be a combination of both a set of partitions from the set of admissible partitions that comprise the DNN and a corresponding allocation vector for the devices that lead to an optimal outcome, indicating that the decision space of the problem can be huge. Therefore, it is necessary to find an efficient solving approach.

Theorem 1 (Problem $\mathcal{P}$ is an NP-Hard Problem): We prove Theorem 1 by identifying $\mathcal{P}$ as an ILP problem and applying the reduction method (detailed in the Appendix). A Linear Programming problem is a kind of optimization toward a linear objective function subject to linear equality or inequality constraints, and the ILP problem is a special case where all optimization variables are integers. The difficulty of solving the optimization problem $\mathcal{P}$ lies in the discreteness of integer variables γ_i and ω_i that describe the partition λ_i. To produce a feasible solution to the problem efficiently, we relax $\mathcal{P}$ by introducing continuous variables $\hat{\gamma}_i$ and $\hat{\omega}_i$ to approximate γ_i and ω_i, respectively. Equation (13) defines the relation between $\hat{\gamma}_i$ and γ_i, where Γ is the input's height and $\hat{\gamma}_i$ describes the proportion that the ith partition covers

$$\gamma_i = \hat{\gamma}_i \Gamma,\ \lambda_i \in \Lambda. \tag{13}$$

Since the input of DNN inference are usually large (e.g., typically of 224 × 224 size from ImageNet data set[1]), the approximation error is minimal. Equation (14) defines the relation between $\hat{\omega}_i$ and ω_i, where L is the number of layers in the DNN model and $\hat{\omega}_i$ describes the proportion of the number of layers that the ith partition covers

$$\omega_i = \hat{\omega}_i |\mathcal{L}|, \ \lambda_i \in \Lambda. \tag{14}$$

Equations (15) and (16) show the numerical constraints for $\hat{\gamma}_i$ and $\hat{\omega}_i$, which are derived from (3) and (4), respectively

$$\sum_{i:\lambda_i \in \Lambda} \hat{\gamma}_i = 1 \ \ \forall l \in \mathcal{L} \tag{15}$$

$$\hat{\omega}_i \leq |\mathcal{L}| \ \ \forall \lambda_i \in \Lambda. \tag{16}$$

The relaxed problem can be solved efficiently (i.e., without having to check all possible solutions as the ILP). However, it can generate infeasible solutions (e.g., noninteger number of rows and layers). Therefore, DISNET gradually narrows down the selection of participating devices (i.e., the devices assigned with the workload) by checking the constraints and iteratively approaching the solution.

There are three key concepts of the DISNET solution; selection of optimal subsets of devices, weighted allocation of model partitions based on the distribution of resources on a subset of devices, and allocation through time. The input of algorithm includes the number of model layers and configuration vectors for model layers, the information about the connectivity of layers defined as the neural network model adjacency matrix, the current computation capabilities of the available devices defined as resource vectors for each IoT device in the IoT network, and the communication capabilities of the available devices defined as the adjacency matrix $\{B_{i,j}^{(k)}\}$ for the network that shows the topology information and the strength of each connection between two device nodes.

Periodically, DISNET needs to update the information on the current computational capabilities of the available devices in the network. To achieve this with less computational complexity, DISNET utilizes a *reference partition* (a section of the neural network model replicated on all the devices with partition height γ_Θ and width ω_Θ). An estimation module uses the reference partition to estimate the device's computing rate in relation to other devices (lines 4 to 6 in Algorithm 1). For each reference partition, the estimation module determines the mean time needed to execute the partition by averaging the time for each of the benchmarking propagations. In most cases, it is not necessary to update at every time slot depending on the dynamicity of the computing resources. Therefore, we assume that the estimation module performs a computation benchmark for IoT devices, every M_D time slot, and sets all the estimates for all time slots between two consecutive benchmarks to the same values. Thus, M_D represents the IoT devices computation benchmark interval.

DISNET takes into account the input node and the output node of the inference operation. We assume the output of the inference operation is required at an output node reachable in the IoT network. The output node could be a node that is connected to an access point or gateway for an external network, or it is directly connected to some actuators, e.g., for a closed-loop system, etc. DISNET identifies the optimal path with the highest average resources between the source and the

[1]https://www.image-net.org/download.php

Algorithm 1: Distributed Micro-Split Deep Learning

Input:
$\Phi = \{1, \ldots, \Phi\}$: Available IoT devices
$A^{(k)} = \{B_{i,j}^{(k)}\}$: Adjacency matrix for network devices
$(\rho, P_c, P_t)(\phi) \ \forall \phi \in \Phi$: Resources vector for device ϕ
$\mathcal{L} = \{1, \ldots, L\}$: Layers for the DNN model
$(\psi, s, c_{in}, c_{out}, p, \Gamma)_l \ \forall l \in \mathcal{L}$: Configuration vectors for model layers
$\mu : \mathbb{R} \to \mathbb{N}, \vartheta(x) = \lfloor x/Q \rfloor$: discretizes throughput in equal intervals of Q bit/s
$\vartheta \in \mathbb{N}$: throughput measurement window size
M_D: devices computation benchmark intervals
$\alpha \in [0, 1]$: energy sensitivity coefficient
Output:
$(\Lambda, a) \in \Omega \times \Phi^{|\Lambda|}$: Optimal partition and allocation sets

```
/* Initial throughput matrix                                   */
b^(1) ← Q
B^(0) ← −Q
/* Loops for each epoch k                                      */
for k ∈ ℕ do
    if k mod M_D = 1 then
        /* Devices computation rate                            */
        foreach φ ∈ Φ do
            ρ(φ) ← (r_i / δ^(k)_{c,i}(φ)) · (ω_i / ω_Θ), λ_i : (γ_Θ, ω_Θ)
    foreach (i, j) ∈ ξ do
        if k > 1 then
            b^(k)_{i,j} ← V_{i,j} / (2δ^(k)_{i,j,λ}(φ)) + V_{j,i} / (2δ^(k)_{j,i,λ}(φ))
        /* Rolling window throughput estimations               */
        q ← max{1, k − ϑ + 1}
        B^(k)_{i,j} ← (1/(k−q+1)) Σ_{a=q}^{k} b^(a)_{i,j}
    /* Start node and end node path                            */
    Ξ ← SelectPath⟨φ_I, φ_O⟩
    L ← LayerModelBlock {1, ..., L}
    foreach W ∈ L do
        foreach φ ∈ Ξ ∪ N(φ) do
            /* Effective computation rate and throughput       */
            R_T(φ) ← α · ρ(φ) + (1 − α) · ζ_c / P_c(φ) +
            (α / N(φ)) Σ_{a∈N(φ)} B^(k)_{a,φ} + (1 − α) · ζ_t / P_t(φ)
        Rank R_T(φ) in descending order by effective rates
        /* Optimal devices subset                              */
        A_T ← {φ_1}, R(φ_1) ∈ R_T(φ)
        Compute {Δ^(0), E^(0)}
        repeat
            τ ← τ + 1
            A'_T ← A'_T ∪ {φ_τ}
            foreach φ ∈ A'_T do
                γ_i ← ρ(φ) / Σ_{φ∈A_T} ρ(φ) Γ, a_φ ∈ a
                ω_i ← W(L), a_φ ∈ a, λ_i ∈ Λ
            Compute {Δ^(τ), E^(τ)}
        until Δ^(τ) ≥ Δ^(τ−1) or E^(τ) ≥ E^(τ−1)
```

output node. DISNET then selects a path with the highest average available resources between the source input node device and the output device. To achieve this, DISNET considers the total device node weights (computation rates) and the edge transmission weights of all nodes on the path and their neighbors to determine an amortized path weight (line 12). This accounts for the differences in the ranges between the device's computational and transmission rates.

To reduce the number of iterations based on the model depth (number of layers) and recalculation of intermediate padding data, DISNET groups the models into similar multiple blocks. Typically a model layer can be grouped into multiple layer blocks of equal computation load. Other studies have determined the blocks according to a user-specified parameter and corresponding cut points [28]. DISNET utilizes an initial model benchmark approach to group sets of similar layers into blocks that can be equally processed in parallel. As a benchmark, DISNET identifies blocks of similar layers based on their configuration vectors that include the input height, kernel size, and padding information.

For a set of layers in a block, DISNET splits the computation load among an *optimal subset* of neighbors within the selected path and neighbors. For effective parallelism, the set of layers needs to be adjusted such that they are all completed at approximately the same time to enhance synchronization. Moreover, it is necessary to decide which devices should participate in the computation task of a set of layers in order to balance the tradeoff between communication and computation time. This is because using more devices can reduce the computation time; however, it can increase the communication overhead, which impacts both the communication time and energy consumption required to exchange data.

DISNET estimates the effective computation rates of all devices in a selected neighborhood and then ranks them in descending order (lines 14 to 17). The effective computation rate takes into account the computation rate of a device node and the communication cost of transmitting the intermediate data needed for the computation of a partition. Depending on the energy sensitivity coefficient of the optimization, the order of ranking the devices varies. Using the energy sensitivity coefficient, reduced computation and transmission power consumption values for devices are added to the node computing weights and edge weights, respectively. The value of α determines how the ranking of the device, i.e., as α gets close to zero, the power has more weight, and as α gets close to one, the computation rates get more weight. Afterward, DISNET iteratively adds a device to the *optimal subset* and observes the computing time or energy until the computing time or energy no longer decreases or all devices are added.

To address the computation heterogeneity of device nodes, the partitions assigned to devices should be adjusted in awareness of their available resources. During parallel collaboration, mutual waiting can severely affect parallelism and increase the overall latency. Previous studies have shown that the inference delay of a layer is roughly proportional to its input size on the sliced dimension [24], [28], i.e., proportional to the input height of a partition in this case. The data size on the sliced dimension decreases linearly; it is also safe to assume that the input size on this dimension is approximately proportional to the inference delay for multiple consecutive layers on a device. For DISNET, the input height is equivalent to the partition height γ_i for an arbitrary partition λ_i. Besides, DISNET is flexible as the input size is not only limited to the first layers but any layer within the model where a partition begins. For an optimal subset of devices to compute a set of layers concurrently, DISNET splits the allocation of the subtasks based on the estimated available resource on the devices in a load-balancing manner. To achieve the *weighted allocation* of partitions based on resources, DISNET splits the input dimension of the partition in a ratio of weighted computation and communication rates of each device node to the total aggregated resources of the selected subset (lines 18 to 27).

Additionally, if the kernel size is very large, but the neighboring partition size is very small, the padding range may even be across three or more devices, which could incur high communication overhead and redundant communication. To reduce the communication between devices, some prior works [24], [47] exploit sending redundant data in advance to avoid the padding issue. In DISNET, the padding data communication is minimized by imposing a principle that requires the allocated partition size in the neighboring device to be not smaller than the padding size unless it owns no partition

$$\gamma_i \geq \lfloor \psi_i/2 \rfloor \quad \forall i \in \{1, \ldots, |\Lambda|\}. \tag{17}$$

Determining the weighted allocation of the partitions to devices using a ratio of available resources in the optimal subset of devices provides a solution with regards to the relaxed constraints as defined in (13) and (15) for $\hat{\gamma}_i$. However, this does not always guarantee a feasible solution for the optimization problem with regard to the original constraints defined as defined in (3) and (17). This is because the height γ_i of a partition λ_i must always be an integer, and the final solutions for both ω_i and γ_i must be discrete values. To ascertain the feasibility of the solution, once a weighted allocation of partitions to an optimal subset of layers is completed, DISNET iteratively approximates the γ_i values using (13) while checking the constraint for the total height Γ defined in (15).

In the DISNET model partitioning scheme, a typical partition does not contain all layers or only one layer. The former causes a large amount of redundant computation on a device node, while the latter suffers from frequent synchronization [22], [23]. Empowered by the fine-grained, flexible scheduling mechanism, DISNET allocates the computation of sets of layers via *allocation through time*. DISNET follows the selected path from the source device node toward the output node when determining the optimal distribution of the model execution task. After the allocation of a set of layers into an optimal subset of neighbor devices, the next block of layers is allocated by selecting another optimal subset of devices until all the layers are allocated into partitions (lines 14 to 27).

DISNET selects the next optimal subset of devices for the next set of layers by searching the remaining subsets of neighboring devices on the selected path toward the output device node. The allocation of the partitions from the next block of layers may include the whole or part of the current optimal

subset of devices and other devices. This maintains the computation flow of the model since, logically, the partitions from the next set of layers are executed after the current set of concurrent partitions is executed, even though some or all of the same devices may be reallocated with parts of other partitions, hence the term allocation through time. Otherwise, the following optimal subset of devices to allocate the partitions is selected from another neighborhood of devices that includes nodes along the path toward the output node. It is worth noting that the selection of an optimal subset to allocate a parallel set of layers is selected in the direction toward the output device node. This reduces the search iterations for DISNET to select an optimal subset of devices for the allocation of more partitions.

DISNET is designed to consider the variability of the system context for efficient cooperative execution of the DNN model. A network information module periodically estimates the link throughput $B^{(k)}$ for a time slot k. Using rolling-window averages increases the channel estimations accuracy because observing throughput changes in a window reduces the influence of noisy instantaneous variations of the throughput (lines 7 to 11). Additionally, the network information module gathers information on the topology of the network of IoT devices. For all devices in the IoT network, if any benchmark has been performed during the current time slots, or the throughput state or the network's topology has changed, the optimal partitions are recomputed; otherwise, the old partitions are used. Similarly, when the input and output device node is changed, DISNET recomputes the partition allocation for efficient inference operations in the new context.

Determining the computing rates is carried out separately for each device and takes a constant number of operations. Determining the network throughput between the devices has the time complexity $\mathcal{O}(|\xi|)$. Identifying blocks of similar layers based on their configuration vectors has the time complexity $\mathcal{O}(|\mathcal{L}|)$ as it requires a single list traversal. Creating the effective computing rates list $R_T(\phi)$ requires $\mathcal{O}(|\Phi|)$ time and sorting $\mathcal{O}(|\Phi|\log|\Phi|)$. Since the operation is carried out iteratively over various sets of layers bounded by $|\mathcal{L}|$, the operation has time complexity $\mathcal{O}(|\mathcal{L}|\cdot|\Phi|\log|\Phi|)$. The outer iteration for adding devices to the optimal subset runs $|\Phi|$ times in the worst case, and determining the time and energy values has the complexity $\mathcal{O}(|\Phi|+|\xi|)$ (utilizing the topological sort for DAG). Since the operation is carried out iteratively over various sets of layers bounded by $|\mathcal{L}|$ and on a subset of devices bounded by $|\Phi|$, the operation has time complexity $\mathcal{O}(|\mathcal{L}|\cdot|\Phi|\cdot(|\Phi|+|\xi|))$. Therefore, DISNET's complexity is upper bounded by $\mathcal{O}(|\mathcal{L}|\cdot|\Phi|^2)$ or $\mathcal{O}(|\mathcal{L}|\cdot|\Phi|\cdot|\xi|)$ for the case $|\Phi|>|\xi|$ or $|\xi|>|\Phi|$, respectively.

Utilizing the relaxed problem and directly determining the γ_i and ω_i for a set of layers on a selected optimal devices subset based on the distribution of their resources requires a small number of operations. Thus, DISNET significantly reduces the work required in searching the space $\forall(\Lambda,a)\in\Omega\times\Phi^{|\Lambda|}$ in determining the γ_i and ω_i values and the allocation vector. Moreover, utilizing the DAG properties on the partitions graph, $G_\Lambda=(\Lambda,\epsilon)$ enhances the algorithm efficiency.

IV. Evaluation

We implemented DISNET[2] in Python, and the DL framework in DISNET is PyTorch.[3] In this section, we compare DISNET against state-of-the-art distributed DNN inference frameworks to validate its performance.

A. Experiment Setup

The key metrics of the evaluation are end-to-end inference latency, energy consumption, and accuracy. The total inference time is calculated by summing all the communication time of exchanging intermediate data and the realistically measured computation time of all the model partitions between the start and the end partitions, as described in (10). The inference energy consumption is the total computation and communication energy of all the devices involved in processing model partitions and exchanging the associated data, as described in (11). The accuracy is determined as a ratio between correct predictions and total predictions of a pretrained model over a set of input data.

We compare DISNET with two state-of-the-art methods MoDNN [23], and DeepSlicing [28]. MoDNN uses a layer-wise parallelization, and it partitions a model at each layer. After completing the computation for every layer, all secondary devices send their suboutputs to the primary server, which merges the suboutputs into a new tensor, then partitions it again into subinputs, distributes them to the secondary devices, and lets them proceed with the next layer. DeepSlicing empirically partitions a DNN model into a set number of fused blocks with approximately the same number of layers. In our experiments, we set the partitioning point.

We implemented the DISNET prototype with the visual geometry group (VGG) as a test model. VGG is widely adopted and evaluated in IoT applications based on edge intelligence. These include image classification in Industrial IoT [48], smart security [5], smart vehicles [7], edge-based speech commands processing, and smart health applications [19]. DISNET models a DNN as a DAG of independent partitions and not as an indivisible executable unit: thus, our approach is generalizable and can be applied to any other DNNs composed of any different arrangement of independently executable partitions layers. This evaluation approach is commonly used in the related literature [19], [27], [49]. First, we tested DISNET on real devices with heterogeneous computing and communication capabilities as a benchmark for power consumption and references for running times. The devices are Raspberry Pi and NVIDIA Jetson Nano with low power settings. Each Jetson Nano is equipped with onboard power sensors, located at the power input of the board, which can be read automatically with the `tegrastats` tool.[4] We employ a plugin power monitor to measure the power consumption of Raspberry Pi.

For flexibility, we conducted the evaluation in a simulated environment. We utilize networkx[5] Python library to

[2]https://github.com/ricsamikwa/DiSNet
[3]https://pytorch.org/
[4]https://github.com/topics/tegrastats
[5]https://networkx.org/

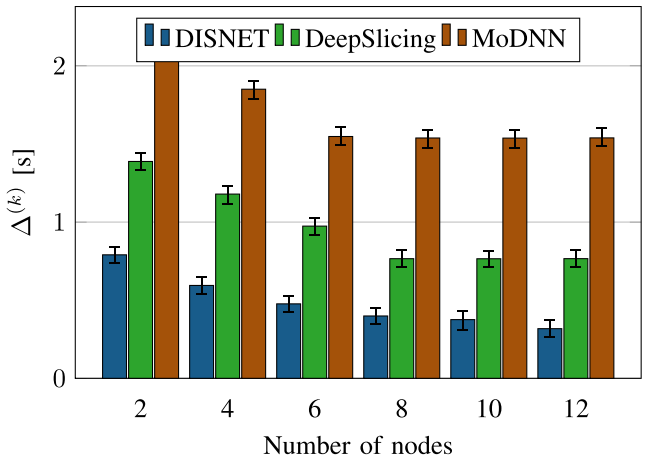

Fig. 6. Comparison of end-to-end inference latency over various device nodes with heterogeneous resources.

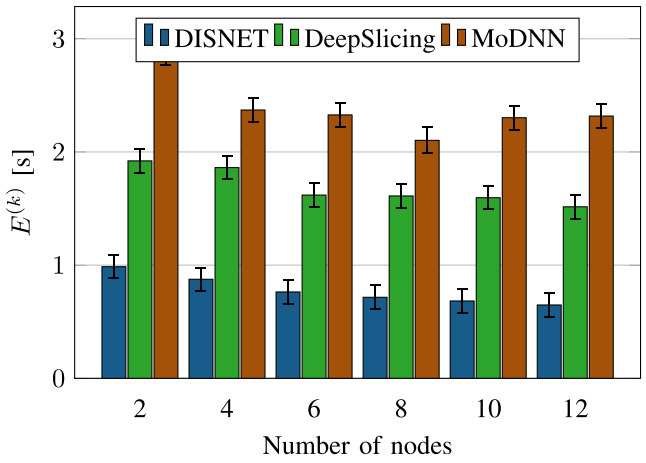

Fig. 7. Comparison of energy consumption for various numbers of device nodes with heterogeneous resources.

generate various networks of heterogeneous devices with variable connectivity and throughput. This is achieved by generating a connected graph for the IoT network with weighted edges representing the throughput $B_{i,j}^{(k)}$ for all connected nodes i and j, and weighted nodes, where the node weight represents the computation rate $\rho(\phi)$ for device ϕ. Assume that the computation rate ρ of a device with the basic resources to process a neural network partition of a fixed height γ_Θ and width ω_Θ is β partitions per second. Since a reference partition is replicated on all devices in the network, we assume the heterogeneity of devices in terms of the computation rates as defined in the set $\{\beta, 2\beta, \ldots, 10\beta\}$. This level of heterogeneity is realizable in a typical IoT network with heterogeneous devices, such as Raspberry Pi A, Raspberry Pi B, and Jetson Nano (with different power settings). Thus, we set the weights of the nodes in the network randomly from the set $\{\beta, 2\beta, \ldots, 10\beta\}$ with a uniform probability.

The links between device nodes in the network are set randomly to obtain a connected graph for the mesh network. Although we are generating the network graphs randomly, we opted for a connected graph as the devices communicate during the DL operations, otherwise, the DISNET solution can run separately. For the edge weights for the network links between devices with random values extracted by a uniform distribution from 5 to 50 Mbit/s. The network throughputs are realizable in an IoT setting as seen from other existing works [46], [49].

B. Inference Acceleration and Energy Consumption Minimization for Variable Numbers of Devices

We evaluate DISNET on various numbers of device nodes with randomly generated device networks. Fig. 6 compares the three schemes (DISNET, DeepSlicing, and MoDNN) on the end-to-end inference latency under different numbers of nodes. The confidence intervals are asymptotic and set at 95% minimum significance level. Generally, it can be seen that as the number of device nodes increases, the computation time decreases. This is because more devices are sharing an inference task. DISNET consistently has the lowest latency and outperforms the other methods because of its adaptability to the model structures and devices' network in terms of computing rates, network throughput, and topologies. DISNET outperforms MoDNN by up to 5.2× and DeepSlicing by up to 2.8× on inference latency. In addition, it can be seen that when the number of devices increases, the speed-up ratio increases. This is because DISNET selects an optimal subset of devices for allocating the sections of concurrent partitions, where the workload allocation size depends on the resources of each device in the subset. Furthermore, besides taking into account the computation and communication capabilities of available devices, DISNET takes into account the network information such as the topology of the network. This allows DISNET to effectively utilize more available devices, even those that are not directly connected to the input device.

Compared to the performance of MoDNN and DeepSlicing, DISNET achieves a higher speedup ratio. This is because although DeepSlicing partitions a model into a set number of fused blocks and can reduce the times of data exchanging between devices, the substantial increase of subinput on each device causes excessive computation and communication overhead. On the other hand, poor adaptability affects MoDNN as it does not take into account the computation and network resources of the available devices jointly. Furthermore, MoDNN mainly suffers from layer-wise synchronization. As a consequence, when the number of workers increases, the latency of MoDNN may increase or remain constant rather than decrease. Both approaches pay less attention to the topology of the networked devices and distribute the inference task to a set of connected devices without exploring the rest of the IoT network.

Fig. 7 compares the three schemes (DISNET, DeepSlicing, and MoDNN) on the energy consumption under different numbers of nodes. The confidence intervals are asymptotic and set at 95% minimum significance level. DISNET consistently has the lowest energy consumption and outperforms the other methods because of its adaptability to the model structures and devices' network resources. DISNET outperforms MoDNN by up to 6× and DeepSlicing by up to 2.5× on inference

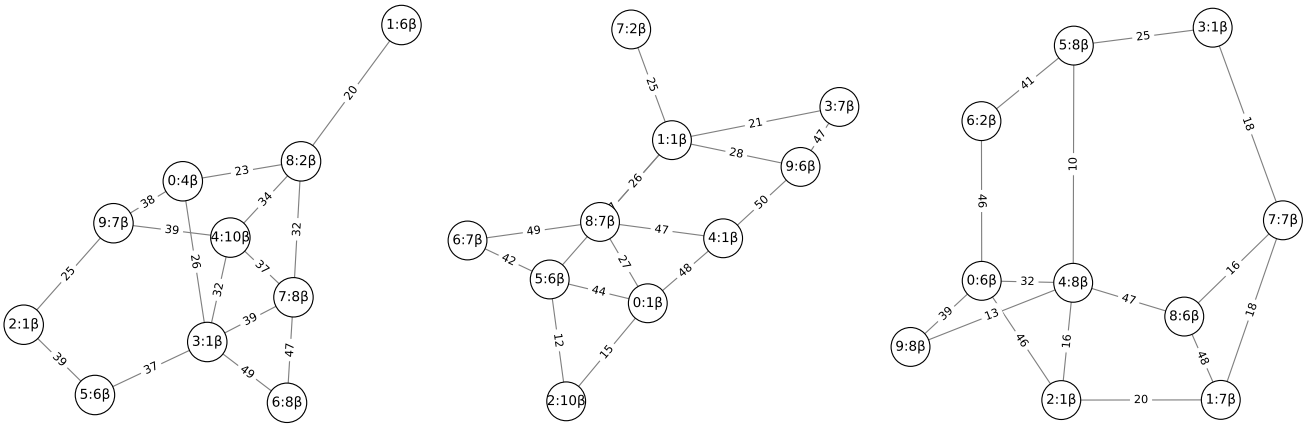

Fig. 8. Mesh networks $\{N1, N2, N3\}$, respectively, with input data sources and corresponding output device nodes.

energy. DISNET selects an optimal subset of devices on which to allocate partitions while giving less weight to the energy-hungry devices, where the workload allocation size depends on the resources of each device in the subset.

Besides taking into account the computation and communication capabilities of available devices, DISNET takes into account the network information such as the topology of the network. This allows DISNET to effectively utilize more available devices even those that are not directly connected to the input device but are less energy hungry. On the other hand, MoDNN and DeepSlicing pay less attention to the energy consumption of the devices selected for the cooperative execution and do not consider the differences in transmission and computation power consumption of devices.

C. Performance Under Variable Multiple Input Device Nodes

In order to evaluate our approach under more realistic application scenarios, we further evaluate the inference latency and overall energy consumption with multiple data sources and a fixed number of devices. In a deployment of an IoT mesh network, for instance as shown in Fig. 3, we assume a possibility for each node to generate the input for the inference operation and it requests cooperation from available connected devices. We assume the output of the inference operation is required at an output node. The output node could be a node that is connected to an access point or a gateway, or it is directly connected to some actuators, e.g., for a closed-loop system. For generalization, we set both the input and output device node numbers to be randomly selected from the set of all the available device nodes with a uniform distribution.

For demonstration, we randomly generated three mesh networks with a fixed number of devices. Fig. 8 shows the generated networks of devices with heterogeneous devices and communication links. As discussed earlier, the weights of the nodes in the network (i.e., device computing rate) are set randomly from the set $\{\beta, 2\beta, \ldots, 10\beta\}$ with a uniform probability. For the edge weights for the network links between devices with random values extracted by a uniform distribution from 5 to 50 Mbit/s. For each network in the set $\{N1, N2, N3\}$, we generate six input and output node pairs for the evaluation. Table III shows the input and output node pair combinations

TABLE III
INPUT DATA SOURCES AND CORRESPONDING OUTPUT NODES

Network	A	B	C	D	E	F
N1	$0 \rightarrow 7$	$1 \rightarrow 0$	$2 \rightarrow 0$	$2 \rightarrow 1$	$4 \rightarrow 5$	$9 \rightarrow 7$
N2	$0 \rightarrow 9$	$4 \rightarrow 1$	$6 \rightarrow 1$	$6 \rightarrow 4$	$6 \rightarrow 8$	$7 \rightarrow 5$
N3	$3 \rightarrow 5$	$5 \rightarrow 8$	$6 \rightarrow 0$	$6 \rightarrow 7$	$9 \rightarrow 2$	$9 \rightarrow 8$

generated from a uniform distribution. We evaluate DISNET in comparisons with DeepSlicing and MoDNN for inference latency and energy consumption for each pair of input and output nodes. For each pair, we run at least 100 inference epochs. The confidence intervals are asymptotic and set at 95% minimum significance level.

Fig. 9 shows the inference latency of for the networks $N \in \{N1, N2, N3\}$, respectively. DISNET outperforms MoDNN by up to 5.2× and DeepSlicing by up to 2.8× on inference latency. This is achieved because DISNET selects an optimal set of devices on which to allocate the partitions, executed in parallel, and where the workload allocation size depends on the resources of each device in the subset. Furthermore, besides taking into account the computation and communication capabilities of available devices, DISNET takes into account the network information, such as the topology of the network. This allows DISNET to effectively utilize more available devices, even those that are not directly connected to the input device. On the other hand, MoDNN and DeepSlicing approaches pay less attention to the topology of the networked devices and distribute the inference task to a set of connected devices.

Fig. 10 shows the energy consumption of the networks $N \in \{N1, N2, N3\}$, respectively. DISNET outperforms MoDNN by up to 6× and DeepSlicing by up to 2.5× on inference latency. This is because DISNET selects an optimal subset of devices for allocating the sections of partitions while giving less weight to the energy-intensive devices, where the workload allocation size is dependent on the resources of each device in the subset. Fig. 11 shows aggregated inference latency and energy consumption under variable multiple input and output device pairs.

Besides taking into account the computation and communication capabilities of available devices, DISNET takes into account the network information such as the

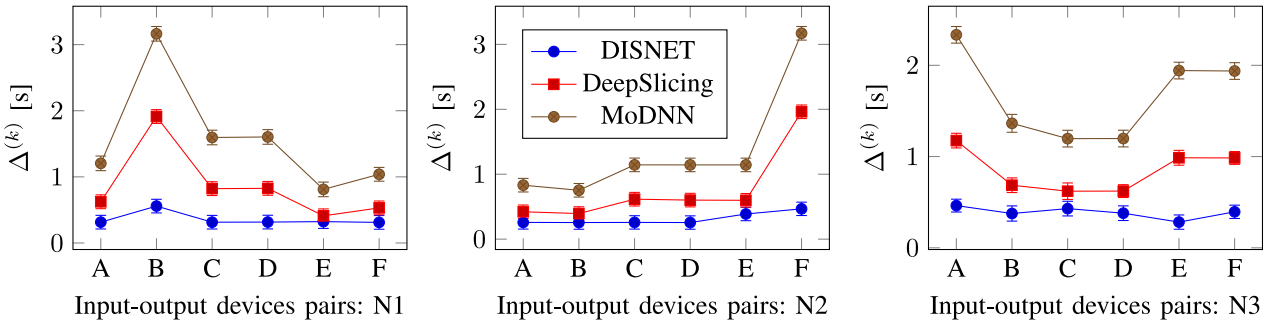

Fig. 9. Inference latency for VGG on mesh networks $\{N1, N2, N3\}$ with input data sources and corresponding output device nodes.

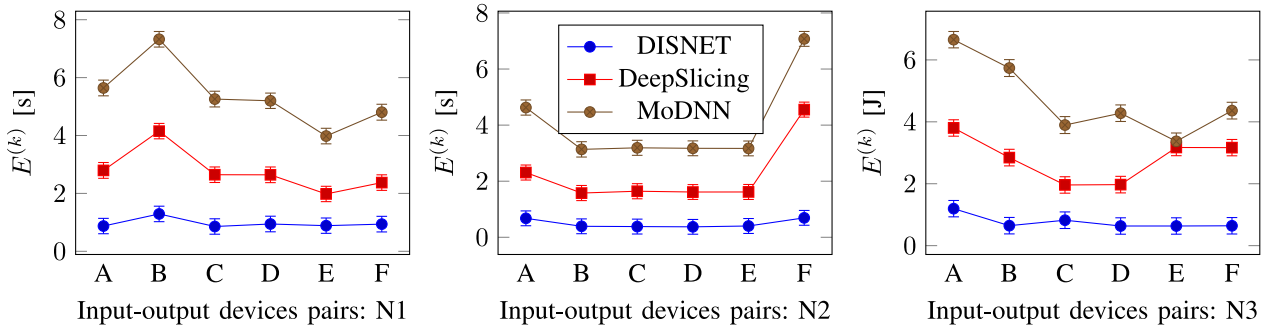

Fig. 10. Energy consumption for VGG on mesh networks $\{N1, N2, N3\}$ with input data sources and corresponding output device nodes.

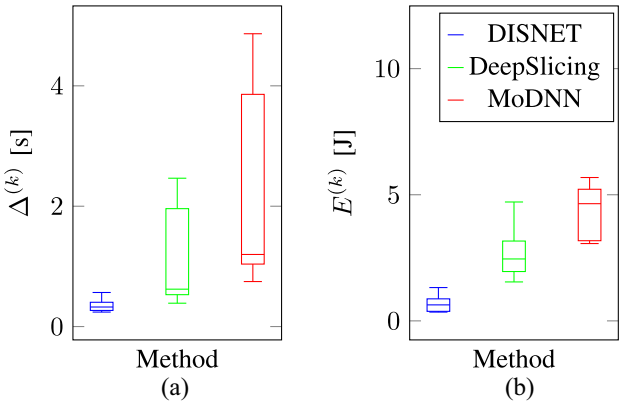

Fig. 11. Aggregated inference latency and energy consumption under variable multiple input data sources and output device nodes. (a) Inference latency. (b) Energy consumption.

topology of the network. This allows DISNET to effectively utilize more available devices even those that are not directly connected to the input device but are less energy hungry. On the other hand, MoDNN and DeepSlicing pay less attention to the energy consumption of the devices selected for the cooperative execution and do not consider the differences in transmission and computation power consumption of devices.

D. Impact of Variable Network Throughput Between Devices

In this experiment, we evaluate the system performance of the different schemes with varying bandwidth. We run the three schemes (DISNET, DeepSlicing, and MoDNN) to investigate how their performance changes according to variations in link throughput between devices over time. Therefore, we modify the network graph generation to set the network maximum throughput between devices, for every round k, to a random value extracted by a uniform distribution from 10 to 45 Mbit/s during the model execution. This means for every round k the edge weights of the generated network are generated randomly between 0 and the set maximum $B^{(k)}$. The DISNET energy sensitivity coefficient is set at $\alpha = 0.5$. For every set value of the maximum throughput $B^{(k)}$, the three schemes are computed for 50 inference epochs while observing the inference latency and energy consumption.

Fig. 12 shows the impact of the system context on the training time and energy consumption of IoT devices at each inference epoch k. DISNET accelerates inference time and minimizes energy consumption up to 3× and 6×, respectively, compared to DeepSlicing and MoDNN. As the bandwidth changes, all three approaches vary their performance. The performance variation comes from two reasons. On the one hand, the communication overhead for necessary data exchange depends on the network conditions. On the other hand, for DISNET, diverse bandwidth values yield diverse partitioning plans, therefore, impacting the performance of the cooperative DNN execution. DISNET consistently outperforms DeepSlicing and MoDNN on inference latency and energy consumption. Fig. 13 shows the probability distributions for inference latency and energy consumption under variable max throughput $B^{(k)}$ set from a uniform distribution between 10 and 45 Mbit/s.

E. Comparison of Model Accuracy

Model accuracy is a key aspect of every intelligent IoT application. Hence, the optimization methods need to consider

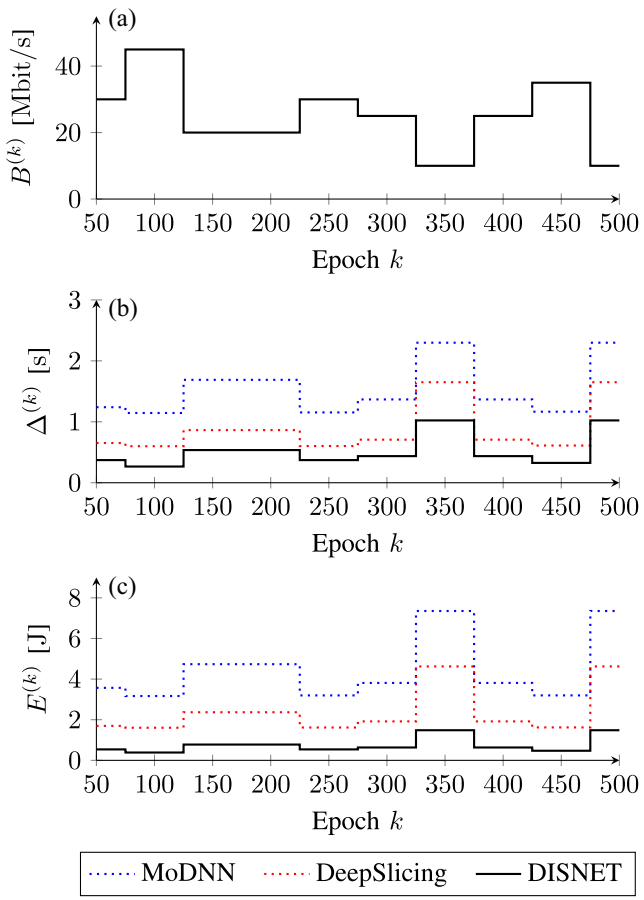

Fig. 12. Impact of the system context, (a) variable max throughput, on (b) inference latency $\Delta^{(k)}$ and (c) energy consumption $E^{(k)}$ for the VGG model at each epoch k.

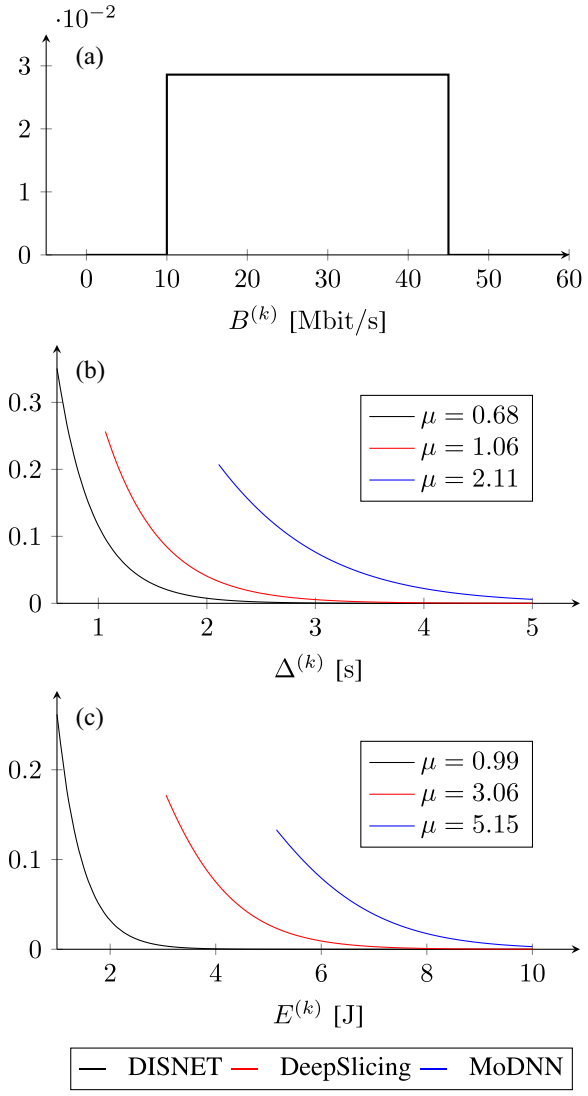

Fig. 13. Distributions for (b) inference latency and (c) energy consumption under variable max throughput $B^{(k)}$ set from a uniform distribution (a) between 10 and 45 Mbit/s.

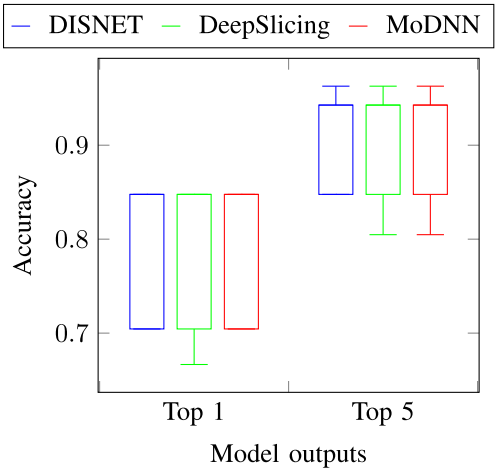

Fig. 14. Inference accuracy using different cooperative DNN execution methods.

the prediction accuracy of the model. We monitored the resulting test accuracy VGG model for DISNET and other approaches. Fig. 14 shows the top-1 and top-5 inference accuracy of DISNET, MoDNN, and DeepSlicing. DISNET achieves comparable inference accuracies as the pretrained model compared to MoDNN and DeepSlicing. DeepSlicing may have some lower accuracy in other predictions. This is because the method ignores the effect of stride and padding for each layer, thereby causing inaccurate calculations of the DNN model output. Distributed model execution techniques are highly beneficial when they have a less significant impact on the resulting model accuracy since the optimization does not compromise the integrity of the intelligent IoT application.

V. Conclusion

In this article, we present DISNET, a distributed micro-split DL scheme that enables cooperative DNN inference while utilizing the collective computing of heterogeneous, ubiquitous, and decentralized IoT devices. We explore the workflow of cooperative inference and formulate it as a constrained optimization problem, which is NP-hard. To solve it efficiently, we design a workload partitioning algorithm to decide efficient partitioning policy heuristically. By jointly considering computation and communication resources, DISNET can find the optimal workload partitioning plan that minimizes the system energy cost and execution latency. To achieve this, DISNET combines vertical (layer based) and horizontal DNN partitioning to enable efficient and parallel execution of

neural network models. Experimental evaluation in dynamic IoT networks shows the efficacy of DISNET in reducing the DNN inference latency and energy consumption by up to 5.2× and 6×, respectively, compared to two state-of-the-art schemes without loss of inference accuracy. We envision the deployment of cooperative DL systems in environments, such as smart homes, Industrial IoT, the Internet of Medical Things, smart environmental monitoring, etc., wherein the devices are willing to cooperate and share their resources. Moreover, as future work, DISNET can be extended to improve the training efficiency for distributed learning and real-time continuous learning in IoT edge environments.

APPENDIX

PROOF OF THEOREM

Proof: We reduce the parallel machines scheduling problem with makespan minimization $P||C_{\max}$ to a special case of $\mathcal{P}$, where all the power parameters are set as 1. Since the $P||C_{\max}$ problem is NP-hard, $\mathcal{P}$ is at least as hard as the $P||C_{\max}$ problem.

We first identify $\mathcal{P}$ as an integer linear programming problem. Considering the variables partition height γ_i (number of rows covered on the partition input side) and partition width ω_i (number of layers in the partition), the constraints (3)–(5) limit both variables into a range of nonnegative integers. As a subproblem, we pick a special case where we set the partition width ω_i equal to the number of layers in the DNN model $|\mathcal{L}|$. Using γ_i, we can obtain the initial workload allocation on each device by multiplying γ_i and the data size of each row [i.e., the input feature map size of (γ_i, W, c_{in})], given the configuration vector of the layer is known. Since the input feature maps of each layer are the output of the prior layer, we can derive the workload of each layer based on its specific configuration. For example, for convolution operation, given the input feature map partition of size $(\gamma, W, c_{\text{in}})$ (height, width, channels), the output size of partition λ_i is $([\gamma - \psi + 2p/s]+1, [W - \psi + 2p/s]+1, c_{\text{out}})$ (height, width, channels). Therefore, we can express the workload size, determined by the input feature map of size r_i, linearly using γ_i. Similarly for $\delta^{(k)}_{c,i}(\phi)$, $E^{(k)}_{c,i}(\phi)$, $\delta^{(k)}_{x,i}(\phi)$, and $E^{(k)}_{x,i}(\phi)$ according to (6)–(9). Without loss of generality, we can infer that the total time $\Delta^{(k)}(a)$ and energy consumption $E^{(k)}(a)$ are linear with γ_i.

In conclusion, all the expressions that determine $\mathcal{P}$ are either linear functions or integer constraints, indicating that $\mathcal{P}$ is an integer linear programming problem. Let the variables γ_i be the jobs to schedule, and all power parameters be 1; we can reduce the $P||C_{\max}$ problem to $\mathcal{P}$ by recognizing the total computing time and energy in $\mathcal{P}$ as the processing time in $P||C_{\max}$. Since $P||C_{\max}$ problem is NP-hard, then the problem $\mathcal{P}$ is at least NP-hard. ■

Alternative Proof: To show the NP-hardness of the problem $\mathcal{P}$, we here consider another special instance of $\mathcal{P}$, in which: 1) the nodes that can process DNN $\mathcal{L}$ are given in advance as $S_\phi = \{1, \ldots, |S_\phi|\}$; 2) the partition height γ_i is always equal to the height of the partition input layer $\sum_{i:\lambda_i \in \Lambda} \gamma_i(l) = \Gamma_l = \gamma_i(l), \ \forall l \in \mathcal{L}$; and 3) each device ϕ will consume $\delta^{(k)}_{c,j}(\phi)$ time in processing layer L_j of $\mathcal{L}$. For this special case, we can treat S_ϕ to be a set of parallel machines and $\mathcal{L}$ to be the set of jobs with precedence. This special problem instance is thus transformed into determining a partition of $\mathcal{L}$ and a policy that can allocate each resulted segment (i.e., partition) to a machine of S_ϕ while minimizing the total processing time or the energy consumption, by determining the optimal values of ω_i for each partition. This can be described as the workload partition problem (WPP) with the precedence constraint of jobs. The WPP is a proven NP-hard problem [50]. To solve the problem proposed in this paper, we need to determine not only the partition of $\mathcal{L}$ by determining the optimal values of ω_i for each partition but also the γ_i values that define the partition height, and the allocation vector $a = (a_1, \ldots, a_{|\Lambda|}) \in \Phi^{|\Lambda|}$ that can contain all or a subset of the available devices. Therefore, the problem $\mathcal{P}$ is generally at least NP-hard.

REFERENCES

[1] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "Federated learning for resource-constrained IoT devices: Panoramas and state-of-the-art," 2020, *arXiv:2002.10610*.

[2] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *J. Biomed. Inform.*, vol. 99, Nov. 2019, Art. no. 103291.

[3] G. Fabregat, J. A. Belloch, J. M. Badía, and M. Cobos, "Design and implementation of acoustic source localization on a low-cost IoT edge platform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3547–3551, Dec. 2020.

[4] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the Industrial Internet of Things," *IEEE Netw.*, vol. 33, no. 5, pp. 96–103, Sep./Oct. 2019.

[5] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4921–4934, Jun. 2019.

[6] H. Song, J. Bai, Y. Yi, J. Wu, and L. Liu, "Artificial intelligence enabled Internet of Things: Network architecture and spectrum access," *IEEE Comput. Intell. Mag.*, vol. 15, no. 1, pp. 44–51, Feb. 2020.

[7] Z. Bi, L. Yu, H. Gao, P. Zhou, and H. Yao, "Improved VGG model-based efficient traffic sign recognition for safe driving in 5G scenarios," *Int. J. Mach. Learn. Cybern.*, vol. 12, no. 11, pp. 3069–3080, 2021.

[8] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[9] Z. Qu, C. Liu, J. Guo, and L. Thiele, "Deep partial updating,"2020, *arXiv:2007.03071*.

[10] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. 57th Annu. Meetinge Assoc. Comput. Linguist.*, 2019, pp. 3645–3650.

[11] J. Henkel, S. Pagani, H. Amrouch, L. Bauer, and F. Samie, "Ultra-low power and dependability for IoT devices (invited paper for IoT technologies)," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 954–959.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[13] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[14] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.

[15] R. Marino, C. Wisultschew, A. Otero, J. M. Lanza-Gutierrez, J. Portilla, and E. de la Torre, "A machine-learning-based distributed system for fault diagnosis with scalable detection quality in industrial IoT," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4339–4352, Mar. 2021.

[16] A. F. R. Neto, F. C. Delicato, T. V. Batista, and P. F. Pires, "Distributed machine learning for IoT applications in the fog," in *Fog Computing: Theory and Practice: Theory and Practice*. London, U.K.: Wiley Telecom, 2020, pp. 309–345.

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

[18] Z. Qu, Z. Zhou, Y. Cheng, and L. Thiele, "Adaptive loss-aware quantization for multi-bit networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 7988–7997.

[19] Y. Gao et al., "Evaluation and optimization of distributed machine learning techniques for Internet of Things," 2021, *arXiv:2103.02762*.

[20] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9241–9254, Oct. 2020.

[21] 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.*, 2019, pp. 1423–1431.

[22] 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.

[23] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "MoDNN: Local distributed mobile computing system for deep neural network," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 1396–1401.

[24] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[25] L. Lockhart, P. Harvey, P. Imai, P. Willis, and B. Varghese, "Scission: Performance-driven and context-aware cloud-edge distribution of deep neural networks," 2020, *arXiv:2008.03523v1*.

[26] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[27] M. Krouka, A. Elgabli, C. B. Issaid, and M. Bennis, "Energy-efficient model compression and splitting for collaborative inference over time-varying channels," 2021, *arXiv:2106.00995*.

[28] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "DeepSlicing: Collaborative and adaptive CNN inference with low latency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2175–2187, Sep. 2021.

[29] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Toward collaborative inferencing of deep neural networks on Internet-of-Things devices," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4950–4960, Jun. 2020.

[30] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

[31] E. Baccarelli, M. Scarpiniti, A. Momenzadeh, and S. S. Ahrabi, "Learning-in-the-fog (LiFo): Deep learning meets fog computing for the minimum-energy distributed early-exit of inference in delay-critical IoT realms," *IEEE Access*, vol. 9, pp. 25716–25757, 2021.

[32] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "Bert loses patience: Fast and robust inference with early exit," 2020, *arXiv:2006.04152*.

[33] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, 2016, pp. 2464–2469.

[34] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic progressive inference of neural networks over device and cloud," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–15.

[35] M. Xue, H. Wu, R. Li, M. Xu, and P. Jiao, "EosDNN: AN efficient offloading scheme for DNN inference acceleration in local-edge-cloud collaborative environments," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 1, pp. 248–264, Mar. 2022.

[36] 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./Apr. 2022.

[37] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[38] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 535–547.

[39] U. Köster et al., "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–14.

[40] N. D. Pham, A. Abuadbba, Y. Gao, T. K. Phan, and N. Chilamkurti, "Binarizing split learning for data privacy enhancement and computation reduction," 2022, *arXiv:2206.04864*.

[41] E. Samikwa, A. Di Maio, and T. Braun, "Adaptive early exit of computation for energy-efficient and low-latency machine learning over IoT networks," in *Proc. IEEE 19th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2022, pp. 200–206.

[42] N. Li, A. Iosifidis, and Q. Zhang, "Collaborative edge computing for distributed CNN inference acceleration using receptive field-based segmentation," *Comput. Netw.*, vol. 214, Sep. 2022, Art. no. 109150.

[43] S. Holly, A. Wendt, and M. Lechner, "Profiling energy consumption of deep neural networks on Nvidia Jetson Nano," in *Proc. 11th Int. Green Sustain. Comput. Workshops (IGSC)*, 2020, pp. 1–6.

[44] C. F. Rodrigues, G. Riley, and M. Luján, "SyNERGY: An energy measurement and prediction framework for convolutional neural networks on Jetson TX1," in *Proc. Int. Conf. Parallel Distrib. Process. Techn. Appl. (PDPTA)*, 2018, pp. 375–382.

[45] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, "Performance analysis of real-time DNN inference on raspberry PI," in *Proc. Real-Time Image Video Process.*, 2018, pp. 115–123.

[46] E. Samikwa, A. Di Maio, and T. Braun, "ARES: Adaptive resource-aware split learning for Internet of Things," *Comput. Netw.*, vol. 218, Dec. 2022, Art. no. 109380.

[47] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, 2019, pp. 195–208.

[48] X. Liu, W. Yu, F. Liang, D. Griffith, and N. Golmie, "Toward deep transfer learning in Industrial Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12163–12175, Aug. 2021.

[49] D. Wu, R. Ullah, P. Harvey, P. Kilpatrick, I. Spence, and B. Varghese, "FedAdapt: Adaptive offloading for IoT devices in federated learning," 2021, *arXiv:2107.04271*.

[50] L. Hu, G. Sun, and Y. Ren, "CoEdge: Exploiting the edge-cloud collaboration for faster deep learning," *IEEE Access*, vol. 8, pp. 100533–100541, 2020.

Eric Samikwa received the M.Sc. degree in computer science and engineering from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2020. He is currently pursuing the Ph.D. degree with the Communication and Distributed Systems Group, Institute of Computer Science, University of Bern, Bern, Switzerland.

His research interests are in the areas of distributed machine learning, federated learning, split learning, edge computing, and Internet of Things.

Antonio Di Maio received the Ph.D. degree in computer engineering from the University of Luxembourg, Luxembourg City, Luxembourg, in 2020, with a thesis on routing and content dissemination in software-defined vehicular networks.

He is a Postdoctoral Researcher of Mobile Networks with the Communication and Distributed Systems Group, University of Bern, Bern, Switzerland. His current research interests fall within the areas of network modeling, scheduling, routing, and channel access.

Torsten Braun (Member, IEEE) received the Ph.D. degree from the University of Karlsruhe, Karlsruhe, Germany, in 1993.

He has been a Full Professor with the Institute of Computer Science, University of Bern, Bern, Switzerland, since 1998. From 1994 to 1995, he was a Guest Scientist with INRIA Sophia-Antipolis, Valbonne, France. From 1995 to 1997, he worked with the IBM European Networking Centre Heidelberg, Heidelberg, Germany, as a Project Leader and a Senior Consultant. He was a Vice President of the SWITCH (Swiss Research and Education Network Provider) Foundation from 2011 to 2019. He was a Director of the Institute of Computer Science and Applied Mathematics, University of Bern from 2007 to 2011, and from 2019 to 2021.

Prof. Braun received the Best Paper Awards from LCN 2001, WWIC 2007, EE-LSDS 2013, WMNC 2014, and the ARMS-CC-2014 Workshop, as well as the GI-KuVS Communications Software Award in 2009. He also received several Best Poster Awards at Bern Data Science Day 2022 and 2021 and the Best Poster Award from Adhoc-Now 2019. In the scope of EU-funded projects, he was leading WPs of FP6-EUQOS and FP7-MCN. Moreover, he coordinated national projects, such as SNSF Swiss Sense Synergy and SNSF CONTACT.