PDD: Partitioning DAG-Topology DNNs for Streaming Tasks

Liantao Wu, *Member, IEEE*, Guoliang Gao, Jing Yu, Fangtong Zhou, *Graduate Student Member, IEEE*, Yang Yang, and Tengfei Wang, *Member, IEEE*

***Abstract*—To enable the inference of high-precision deep neural networks (DNNs) on resource-constrained devices, DNN offloading has been widely explored in recent years. Some works have also integrated the chain-topology DNN (CDNN) offloading with pipeline processing to further reduce inference delay when processing streaming tasks. To improve the accuracy of the inference results, the topology of DNN tends to evolve from chain topology to directed acyclic graph (DAG) topology. However, most of the existing works do not study partitioning and offloading DAG-topology DNNs (DDNNs) for streaming tasks. Moreover, when partitioning computationally expensive DNN models, multipartitioning probably outperforms the bi-partitioning method, and most of the works do not study multipartitioning DAG-topology DNNs. In this article, we propose a more general multipartitioning and offloading method for large-scale DDNNs to process streaming tasks, which can adaptively partition DDNNs into multiple parts considering the computing power and bandwidth of all available computing units. Specifically, we first present a transforming method based on topological sorting that can losslessly transform DAG-topology DNNs into CDNNs. Then, based on greedy and dichotomy ideas, a multipartitioning algorithm is designed to partition and offload CDNNs. In this way, we can solve DDNNs' multipartitioning problem based on the proposed transforming and partitioning algorithms. Experiments show that the method proposed in this article significantly outperforms bi-partitioning and nonpartitioning methods when offloading computationally expensive DNN models (https://github.com/sreasearcher/PDD-Code).**

***Index Terms*—Directed acyclic graph (DAG) topology, deep neural network (DNN) partitioning, streaming tasks.**

Manuscript received 28 July 2023; revised 13 September 2023; accepted 26 September 2023. Date of publication 17 October 2023; date of current version 7 March 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4300300; in part by the National Natural Science Foundation of China (NSFC) under Grant 62202307 and Grant 52101403; and in part by the National Natural Science Foundation of China (NSFC) Key Project Program under Grant 61932014. *(Corresponding author: Tengfei Wang.)*

Liantao Wu is with the Software Engineering Institute, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China (e-mail: wult@shanghaitech.edu.cn).

Guoliang Gao, Jing Yu, and Fangtong Zhou are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China (e-mail: gaogl@shanghaitech.edu.cn; yujing2@shanghaitech.edu.cn; zhouft@shanghaitech.edu.cn).

Yang Yang is with the IoT Thrust, Hong Kong University of Science and Technology (Guangzhou), Guangzhou 510000, China (e-mail: yyiot@hkust-gz.edu.cn).

Tengfei Wang is with the State Key Laboratory of Maritime Technology and Safety and the School of Transportation and Logistics Engineering, Wuhan University of Technology, Wuhan 430063, China (e-mail: wangtengfei@whut.edu.cn).

Digital Object Identifier 10.1109/JIOT.2023.3323520

I. Introduction

AS AN ESSENTIAL tool for deep learning, deep neural networks (DNNs) have been applied to support various practical applications, such as face recognition, video analysis, and natural language processing. Deeper (i.e., more layers) and larger (i.e., more neurons per layer) DNNs are becoming increasingly popular, as they can achieve higher precision performance [1]. For example, all the champion DNNs of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) after 2015 have more than 100 layers. In particular, YOLOv4 has more than 400 layers [2].

Due to outstanding performance of artificial intelligence, DNNs have been widely used in various mobile intelligent applications, such as Apple Siri and Google Assistant etc., which run on mobile devices, such as smartphones and tablets. These intelligent applications have huge demands for computing power [3], while mobile devices only have limited computing power. With delay constraints [4], it is challenging for many practical deep learning algorithms to be deployed on devices [5].

There are mainly two kinds of approaches to deploy a DNN model on mobile devices.

1) Leveraging the model compression mechanism to reduce the model accuracy for less computation [6].
2) Deploying DNN models in a distributed manner by offloading part or all the computation tasks to other computing units to reduce the local workload.

Model compression methods are model specific, which highly depend on the DNN architecture. The accuracy of a compressed model is also not guaranteed theoretically [7]. Therefore, DNN offloading has become a research hot spot in recent years to deploy a DNN model on mobile devices [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]. DNN offloading refers to offloading the DNN tasks of user equipments (UEs) to computing units (CUs), which execute the tasks and return the computation results to UEs. The CUs generally possess more computing resources than UEs, so the computation delay of the task can be reduced by offloading. However, DNN offloading requires UEs to transmit DNN parameters together with the input data to CUs, which induces additional communication delay compared with local computing. Thus, it is essential to balance the computation and communication delay to minimize the overall delay.

Traditionally, the CU for offloading was usually the cloud because of its abundant resources [9]. However, UEs generally connect to the cloud through a wide area network (WAN), which may involve a considerable communication delay [23]. With the development of edge computing, edge nodes become another offloading preference. They enable a smaller communication delay but introduce more computation delay as edge nodes have limited computing resources compared with the cloud. Therefore, the focus becomes balancing the computation and communication in the scenarios of edge computing [10], [11], [12], [13], [14] and edge-cloud computing [15], [16], [17], [18], [19], [20], [21]. Note that DNN partitioning does not change the overall structure of the DNN, thus it does not affect the inference performance and the methods that improve the model robustness and trustworthiness during DNN inference [24].

Early works partition the DNN into two parts, which are deployed on different devices to enable collaborative inference. Li et al. [25] proposed a framework for device-edge collaboration in DNN inference, adapting the computation partitioning between devices and the edge to leverage the resources of powerful edge devices for real-time DNN inference. Hu et al. [15] proposed a method for splitting more complex and diverse DAG-topology DNNs (DDNNs). They introduced the dynamic adaptive DNN surgery (DADS) strategy based on the minimum-cut algorithm to find an optimal DDNNs splitting approach by balancing computation and transmission time. Li et al. [11] investigated a new problem of maximizing the throughput of DNN inference under delay constraints. They accelerated DNN inference by exploring DNN partition and multithreaded parallel execution to maximize the number of DNN service requests with acceptable latency.

There are also a few works studying multipartitioning method. Zhao et al. [26] introduced a deep Q-network (DQN)-based DNN partitioning strategy to partition the neural network into multiple parts and assign them to multiple drones for collaborative computation. The partitioning strategy aims to minimize the energy consumption of DNN collaborative inference among multi-UAVs within latency and reliability requirements. Ren et al. [27] investigated Device-Edge-Cloud collaborative scenario for layer granularity DNN partitioning with multipartitioning points, where DNN layers will be assigned to the edge or cloud server for inference acceleration while others will be completed on the mobile devices to reduce the system cost.

Another important branch is the DNN offloading for streaming tasks (STs). STs refer to a flow composed of multiple orderly DNN tasks associated with the same DNN but requiring different input data. For example, real-time monitoring systems in enterprises, real-time risk control systems in the financial industry, and real-time video processing required in the live streaming industry all belong to streaming tasks. Research on accelerating the processing of streaming tasks using DNN is very important. In this type of problem, DNN offloading generally occurs only once, while the data transmission and processing occur multiple times. The optimizing object is to minimize the maximum transmitting and processing delays.

To solve this problem, Gao et al. [22] proposed a layer-level offloading model and designed a depth-first search (DFS)-based algorithm with path establishment, calculation and record stages. Yu et al. [28] formulated a noncooperative potential game, proved the existence of the Nash equilibrium (NE) and developed a linear-complexity algorithm to achieve the NE. Gao et al. [29] discussed the thread scheduling involved in ST-oriented DNN offloading, deduced the relationship between the thread blocking and the average task delay (ATD) and prove that an appropriate buffer setting can reduce blocking times.

However, none of the existing works investigated ST-oriented multipartitioning models for DNNs with the directed acyclic graph (DAG) topology, i.e., DDNNs. For computationally expensive DNN models, partitioning DDNNs into multiple parts and offloading them to multiple CUs might achieve a much less ATD than bi-partitioning methods and nonpartitioning methods [30].

Motivated by these facts, we propose a layer-level multipartitioning model in this article, which can adaptively partition DDNNs into multiple parts, considering the computing power and bandwidth of all available computing units. We first present a transforming method based on topological sorting, which can losslessly convert a DDNN to a chain-topology DNN (CDNN). Then, a multipartitioning algorithm for CDNNs is designed based on greedy and dichotomy ideas. We solve DDNNs' multipartitioning by running the transforming and partitioning algorithms sequentially. The function's input is a CDNN part with a group of CUs, while its output is an intermediate offloading strategy. Each recursion redivides the CUs into two groups, repartitions the CDNN part into two parts, offloads them to the CU groups, and then initiates new recursions with the partitioned parts and groups or exits. When all the recursions exit, the obtained "intermediate offloading strategies" will compose a complete. Experimental results show that our proposed multipartitioning method significantly outperforms the bi-partitioning and nonpartitioning methods when offloading various DNN models.

In summary, we make the following contributions in this article.

1) We propose a more general multipartitioning and offloading method for high-precision DAG-topology DNNs to process streaming tasks, which can speed up the cooperative DNN inference.
2) We derive a topological sorting-based DAG-to-chain topology transforming method and design a multipartitioning algorithm, which are used for solving the DDNNs' multipartitioning problem.
3) Extensive experiments are conducted and the results show that our method can significantly reduce the inference latency of various DNN models.

The remainder of this article is organized as follows. Section II explains possibly ambiguous terms. Section III introduces the topological sorting-based transforming method. Section IV presents our multipartitioning algorithm. Section V is the experimentation. Finally, Section VI concludes this article.

II. Preliminaries

For ease of understanding, we interpret the possibly ambiguous terms before starting the modeling. The expounded terms include CU, CDNN, multipartitioning, CDNN part, offloading strategy, intermediate offloading strategy, and partitioning and offloading.

1) CU: A CU refers to a device with computing and communicating capabilities excluding UEs, such as edge and cloud servers.

2) CDNN: A CDNN refers to a DNN with standard connections (i.e., from one layer to its next layer), shortcuts (i.e., from one layer to a nonadjacent layer behind it), and loop connections (i.e., from one layer to itself). The numbers of all the three categories of connections can be zero.

3) Multipartitioning: Multipartitioning refers to partitioning a DNN into multiple parts. It includes bi-partitioning cases (i.e., partitioning a DNN into two parts) and nonpartitioning (i.e., keeping DNNs as they are).

4) CDNN Part: A CDNN part is several adjacent layers of a CDNN. For example, for a CDNN with the layer set $\{1, 2, 3\}$, its CDNN parts are the subsets $\emptyset$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{2, 3\}$ and $\{1, 2, 3\}$. Note that the subset $\{1, 3\}$ is not a CDNN part because layers 1 and 3 are nonadjacent.

5) Offloading Strategy: An offloading strategy indicates how to partition DNNs into several parts and how to offload these parts to available CUs.

6) Intermediate Offloading Strategy: An intermediate offloading strategy refers to the intermediate output of each recursion. It indicates how the input of the recursive function is divided into two CDNN parts and two CU groups and which part should be offloaded to which group.

7) DNN Partitioning and Offloading: DNN partitioning refers to cutting a DNN from some connections to divide it into multiple parts, while offloading refers to placing the partitioned parts on the corresponding CUs. Since our adaptive partitioning model considers CUs' resources, the result exactly includes how the partitioned parts should be offloaded.

Note that DNN partitioning involves partitioning the DNN model into smaller parts that can be offloaded to different CUs. Resources partitioning refers to the allocation and utilization of computing or communication resources for executing part of the DNN [31], it does not consider the DNN partitioning.

III. Topological Sorting of DDNNs

In this section, we first present the DNN model, then introduce how to transform a DDNN to a CDNN based on topological sorting.

A. DNN Model

DNN models usually consist of multiple layers, each with one or more inputs and one or more outputs. The output of each layer is sent to the next layer as the input of the following layer. If we treat each layer in DNN as a vertex, a DNN can be modeled as a DAG. DNNs can always be modeled as DAG-topology DNNs, and DNNs with chain topology are the special cases of DAG-topology DNNs.

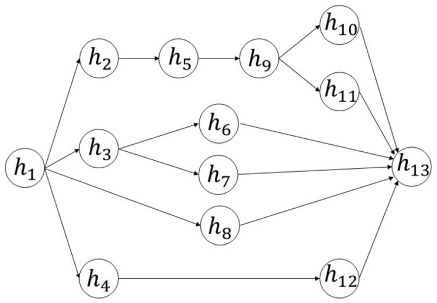


Fig. 1. DAG structure of inception v4 ($\mathcal{I}$) [32].

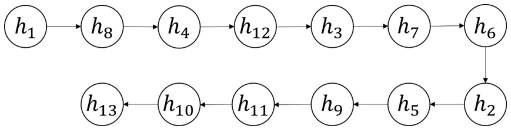


Fig. 2. Topological sorting of $\mathcal{I}$.

Formally, we denote a n-layer DDNN as $G = \{H, L, W\}$, where $H = \{h_i\}, i \in \{1, 2, \ldots, n\}$ is the set of DAG vertices representing DDNNs' layers. L represents all directed edges in the DAG, where $< h_i, h_j > \in L$ indicates that the layer i needs to transmit its output to layer j. W represents the weights of vertices and edges, where $w(h_i) \in W$ represents the computation delay of layer i, $w(< h_i, h_j >) \in W$ represents the amount of data that needs to be transmitted from layer i to layer$j$.

B. Transformation From DDNNs to CDNNs

In graph theory, topological sorting refers to a chained sequence of all vertices of a DAG. The sequence must meet the following two conditions.

1) Each vertex must appear only once.
2) If there is a path from vertex A to vertex B, then vertex A appears before vertex B in the sequence.

Therefore, to achieve the transformation from a DDNN to a CDNN, we first obtain the topological sorting of DDNN to make it a linear sequence, then convert the DDNN connections into standard connections, loop connections, and shortcuts on the linear sequence. This transformation enables offloading models and algorithms suitable for CDNNs to be available for DDNNs' offloading. Next, we will illustrate this process with an example.

Fig. 1 shows the structure of inception v4 ($\mathcal{I}$) [32], which is a complex DAG topology. The DAG's vertices ($h_i, i \in \{1, 2, \ldots, 13\}$) represent $\mathcal{I}$'s layers, and the directed edges represent the connections between the layers. The starting point of an edge needs to transmit its output data to the ending point of the edge. The topological sorting of $\mathcal{I}$ is shown in Fig. 2, which is $h_1 \rightarrow h_8 \rightarrow h_4 \rightarrow h_{12} \rightarrow h_3 \rightarrow h_7 \rightarrow h_6 \rightarrow h_2 \rightarrow h_5 \rightarrow h_9 \rightarrow h_{11} \rightarrow h_{10} \rightarrow h_{13}$. Note that the directed edges in Fig. 2 are only used to specify the vertices' order, and they do not represent layer connections. Next, we match the edges in Fig. 1 and those in Fig. 2, which are denoted as sets $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. For a layer $l \in \mathcal{L}_1 - \mathcal{L}_2$, we add it to $\mathcal{L}_2$. For a layer $l \in \mathcal{L}_2 - \mathcal{L}_1$, we set its weight to 0 (i.e., no data should be passed through this edge). For a layer $l \in \mathcal{L}_1 \cap \mathcal{L}_2$,

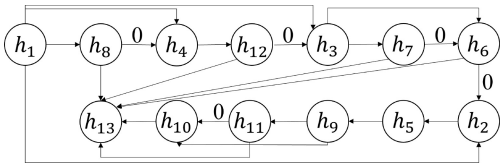


Fig. 3. Chained form of $\mathcal{I}$.

we keep it as it is. Then, we get the structure in Fig. 3. It can be seen that the added edges are all shortcuts. Accordingly, inception v4 is represented as a chain topology with only standard connections and shortcuts so that it can be offloaded by the models/algorithms designed for CDNNs.

We denote the topological sorting of H as an orderly set $H' = \{h'_i\}$ and add directed edges to vertices according to H' to get $L' = \{< h'_i, h'_j >\}, j = i + 1$. In L', the directed edge $< h'_i, h'_j > \in (L' \cap L)$ has the weight $w(< h'_i, h'_j >) \in W$, and the directed edge $< h'_i, h'_j > \in (L' - L)$ has the weight 0. We use W' to denote the set consisting of the weights of vertices in H' and the weights of edges in L'. The weight of vertex h'_i is denoted as $w'(h'_i)$ and the weight of edge $< h'_i, h'_j >$ is denoted as $w'(< h'_i, h'_j >)$. Accordingly, the chained form of G can be denoted as $G' = \{H', L', W'\}$. Specifically, in the example of inception v4, the relationship between H' and H is as follows:

$$\begin{aligned} h'_1 &= h_1, h'_2 = h_8, h'_3 = h_4 \\ h'_4 &= h_{12}, h'_5 = h_3, h'_6 = h_7 \\ h'_7 &= h_6, h'_8 = h_2, h'_9 = h_5 \\ h'_{10} &= h_9, h'_{11} = h_{11}, h'_{12} = h_{10} \\ h'_{13} &= h_{13}. \end{aligned} \tag{1}$$

Note that G' only includes standard connections and shortcuts because of the properties of topological sorting. Recall that if there is a path from vertex A to vertex B, then vertex A appears before vertex B in the sequence. Namely, the connections in G' always point from a vertex to a vertex behind it. If the connected vertices are adjacent, the edge is a standard connection. If not, the edge is a shortcut.

IV. Offloading CDNNs Based on Grouping Recursion

The transforming method proposed in Section III helps to convert complex DDNNs to simple CDNNs, enabling DNN partitioning methods for CDNNs to be applied to DDNNs. Accordingly, in this section, we propose a grouping recursion algorithm for CDNNs to achieve DDNNs' offloading.

We first present the mathematical formulation of the DNN offloading problem. Then we introduce a bi-partitioning algorithm (BPA) that can partition a CDNN into two parts and offload them to two CUs. Based on the proposed BPA, we design the multipartitioning algorithm using the grouping recursion.

A. Problem Formulation

Denote the set of all the CUs as $\mathcal{M}$. We first partition the DNNs into two parts and offload them to two CU groups, denoted as E and C. Specifically, we arrange CUs in ascending order and partition them from the middle. Namely, if there are m arranged CUs, then the first $\lfloor (m/2) \rfloor$ will compose group E, while the rest $m - \lfloor (m/2) \rfloor$ CUs will form group C. The computing power of each group is the sum of the computing power of computing units in the group.

We denote the CUs composing group E and group C as $\mathcal{M}_e$ and $\mathcal{M}_c$, respectively. We have $\mathcal{M}_e \cup \mathcal{M}_c = \mathcal{M}$ and $\mathcal{M}_e \cap \mathcal{M}_c = \emptyset$. Similarly, we denote the set of all CDNN layers as $\mathcal{G}$, the CDNN part offloaded to group E as $\mathcal{G}_e$, and the CDNN part offloaded to group C as $\mathcal{G}_c$, respectively. We have $\mathcal{G}_e \cup \mathcal{G}_c = \mathcal{G}$ and $\mathcal{G}_e \cap \mathcal{G}_c = \emptyset$. Besides, denote F_e and F_c as the computation delay of E and C, respectively.

Since E and C are both CU groups, their computation delay cannot be directly measured. We obtain them as follows. First, we measure the computation amounts of CDNN layers and the computing power of CUs, denoted as $\Psi = \{c_i\}, i \in \{1, 2, \ldots, n\}$ and $\mathcal{R} = \{r_j\}, j \in \{1, 2, \ldots, m\}$, where c_i, r_j, n, and m are the computation amount of layer i, the computing power of CU j, the number of layers, and the number of units, respectively. Then, based on the definition that the computing power of a CU group is the sum of the computing power of all the CUs in the group, F_e and F_c can be calculated with the following:

$$F_e = \frac{\sum_{h_i \in \mathcal{G}_e} c_i}{\sum_{u_j \in \mathcal{M}_e} r_j} \tag{2}$$

$$F_c = \frac{\sum_{h_i \in \mathcal{G}_c} c_i}{\sum_{u_j \in \mathcal{M}_c} r_j}. \tag{3}$$

We define the set of CUs obtained from each bi-partition as u_e and u_c, and the partial results obtained from each binary partition in the CDNN as p_e and p_c. After offloading p_e to one CU group for processing, the intermediate results are transmitted to the other CU group. Let s_i represent the size of the output data for layer i, and $S = \{s_i\}, i \in \{1, 2, \ldots, n\}$ represents the set of output data sizes. Assuming that all CUs and users are in the same local area network (LAN) with the same transmission speed, the bandwidth is represented as b. We define the transmission delay from the E group to the C group as $T^t_{e,c}$, and similarly, the transmission delay from the C group to the E group as $T^t_{c,e}$. The transmission delay for the ith partition is represented as

$$F^t_i = T^t_{e,c} = T^t_{c,e} = \frac{s_i}{b}. \tag{4}$$

When offloading p_e to u_e, the intermediate results of the output are obtained. u_e transfers the intermediate results to u_c, where u_c receives and processes the intermediate results. The transmission and computation can be executed in parallel. Therefore, the average task latency obtained from each binary partition is

$$F = \max\{F_e, F_c, F^t_i\}. \tag{5}$$

Our objective is to minimize the maximum average task latency, i.e.,

$$\min \quad F \tag{6}$$

$$\text{s.t.} \quad F = \max\left\{F_e, F_c, F_i^t\right\} \tag{7}$$

$$F_i^t = \frac{s_i}{b} \tag{8}$$

$$F_e = \frac{\sum_{h_i \in \mathcal{G}_e} c_i}{\sum_{u_j \in \mathcal{M}_e} r_j} \tag{9}$$

$$F_c = \frac{\sum_{h_i \in \mathcal{G}_c} c_i}{\sum_{u_j \in \mathcal{M}_c} r_j} \tag{10}$$

$$1 \le i \le n, 1 \le j \le m. \tag{11}$$

B. Algorithm Design

To solve the above problem, we design the grouping recursion-based multipartitioning algorithm, which consists of two steps.

1) We introduce the BPA, a BPA that can partition a CDNN into two parts and offload them to two CUs.
2) We present the grouping recursion with BPA (GRB), a grouping recursion function with BPA. GRB takes a CDNN part with a group of CUs as input, divides the CUs into two groups, and executes BPA to output intermediate offloading strategies.

Particularly, we set the same bandwidth for all the units, which is common in a LAN. Besides, similar to [33], we ignore the time of CUs transmitting results back to UEs because the size of results is negligible compared to the size of tasks.

Recall that the terms "CDNN part" and intermediate offloading strategy refer to several adjacent layers of a CDNN and the output of the offloading algorithm in each recursion, respectively. In the algorithm, CDNN layers are represented by their computation amounts and data sizes, while CUs are represented by their computing power and bandwidth. Although it is a coding detail, it has been included in our pseudocode. Thus, we explain in advance here.

As for more details, at the beginning of each recursion, GRB checks whether the number of CUs is more than one. If so, the CUs can be divided into two groups so that two new recursions will start with the output of the current recursion (i.e., an intermediate offloading strategy containing two CDNN parts and their corresponding CU groups) as their input. Otherwise, the current recursion will directly exit. When all the recursions exit, the obtained intermediate offloading strategies will exactly compose an offloading strategy. We specify BPA and GRB in Sections IV-B1 and IV-B2, respectively.

1) Bi-Partitioning Algorithm: BPA bi-partitions a CDNN into two parts, which are then offloaded to two CUs. Specifically, a CDNN is partitioned into two parts by cutting the connections starting at certain layer. Then, we choose an appropriate CU for each CDNN part to minimize the maximum step delay.

The involved challenges are selecting the optimal cutting point and offloading strategy. For example, Fig. 4 shows a possible offloading strategy for an 8-layer CDNN, where the CDNN is partitioned at layer 2. Accordingly, layers 1 and 2 compose a CDNN part (p_1), while the other layers (i.e., layers 3-8) compose another part (p_2). Besides, p_1 and p_2 are offloaded to u_1 and u_2, respectively.

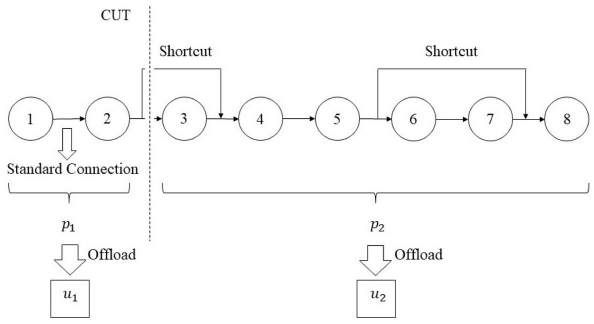


Fig. 4. Example where an 8-layer CDNN is offloaded to two CU.

Formally, to obtain the optimal bi-partitioning and offloading strategy of an n-layer CDNN, we need to iterate all the CDNN layers $h_i \in H$ and repeat the partitioning and offloading process. Accordingly, the algorithm maintains two variables t^* and P^* to record the current optimal ATD (t^*) and the corresponding partitioning and offloading strategy (P^*). Then the variables are altered in the iterations. In the ith iteration, the CDNN is partitioned into two parts (p_1 and p_2) and offloaded to two CUs (u_1 and u_1). For the case where p_1 is offloaded to u_1 (CASE ONE), the corresponding ATD is represented as t_1. Otherwise (CASE TWO) the ATD is t_2. If $t_1 = \min(t_1, t_2) < t^*$, t^* and P^* will be overwritten by t_1 and CASE ONE, respectively. Or if $t_2 = \min(t_1, t_2) < t^*$, t^* and P^* will be overwritten by t_2 and CASE TWO, respectively. Otherwise, the algorithm will do nothing and enter the next iteration. After the iterations, the values recorded by t^* and P^* are the global optimal ATD and the corresponding partitioning and offloading strategy.

The pseudocode of BPA is shown in Algorithm 1, which includes more coding details based on the description above. The input are the parameters of CUs and CDNNs: the computing power and bandwidth of u_1 and u_2, the computation amounts and input/output data sizes of CDNN layers, and the layer number n. The output are the optimal ATD t^* and the corresponding partitioning and offloading strategy P^*. As for the coding logic, BPA first initializes t^* as ∞, avoiding the case where t^* and P^* cannot record anything because t^* is lower than all the obtained ATDs. Next, BPA starts the iteration with a variable i changing from 1 to n. In the ith iteration, the ATDs for CASE ONE (t_1) and CASE TWO (t_2) are calculated sequentially. Then the smaller one of t_1 and t_2 is compared with t^* to determine whether t^* and P^* should be overwritten. Notably, t_1^c, t_2^c, $t_{i,1,2}^t$, and $t_{i,2,1}^t$ are all middle variables, which represent the computation delay of u_1, the computation delay of u_2, the communication delay of h_i's output from u_1 to u_2, and the communication delay of h_i's output from u_2 to u_1, respectively. They can be obtained by the following:

$$t_1^c = \begin{cases} \frac{\sum_{j=1}^{i} c_i}{r_1}, & \text{if CASE ONE} \\ \frac{\sum_{j=i+1}^{n} c_i}{r_2}, & \text{if CASE TWO} \end{cases} \tag{12}$$

Algorithm 1 BPA

Input: The computing power and bandwidth of u_1 and u_2. The computation amounts and input/output data sizes of CDNN layers. The number of CDNN layers n.

Output: The optimal ATD t^* and the corresponding partitioning and offloading strategy P^*.

1: $t^* \leftarrow \infty$;
2: **for** $i = 1 \rightarrow i = n + 1$ **do**
3: **CASE ONE:**
4: $t_1^c \leftarrow$ The sum of computation delay of h_j ($j \in [1, i) \cap N$) on u_1;
5: $t_2^c \leftarrow$ The sum of computation delay of $h_{j'}$ ($j' \in [i, n] \cap N$) on u_2;
6: $t_{i,1,2}^t \leftarrow$ The time transmitting the output of the last DNN layer on u_1 to u_2;
7: $t_1 \leftarrow \max(t_1^c, t_2^c, t_{i,1,2}^t)$;
8: **CASE ONE END**
9: **CASE TWO:**
10: $t_1^c \leftarrow$ The sum of computation delay of $h_{j'}, j' \in [i, n] \cap N$ on u_1;
11: $t_2^c \leftarrow$ The sum of computation delay of $h_j, j \in [1, i) \cap N$ on u_2;
12: $t_{i,2,1}^t \leftarrow$ The time transmitting the output of the last DNN layer on u_2 to u_1;
13: $t_2 \leftarrow \max(t_1^c, t_2^c, t_{i,2,1}^t)$;
14: **CASE TWO END**
15: **if** $t_1 \leq t_2$ and $t_1 < t^*$ **then**
16: $t^* \leftarrow t_1$;
17: $P^* \leftarrow$CASE ONE;
18: **else if** $t_2 < t_1$ and $t_2 < t^*$ **then**
19: $t^* \leftarrow t_2$;
20: $P^* \leftarrow$CASE TWO;
21: **end if**
22: **end for**
23: return $[t^*, P^*]$;

$$t_2^c = \begin{cases} \frac{\sum_{j=i+1}^{n} c_i}{r_1}, & \text{if CASE ONE} \\ \frac{\sum_{j=1}^{i} c_i}{r_2}, & \text{if CASE TWO} \end{cases} \tag{13}$$

$$t_{i,1,2}^t = t_{i,2,1}^t = \frac{s_i}{b} \tag{14}$$

where c_i, r_1, r_2, s_i, and b are the computation amount of h_i, the computing power of h_1, the computing power of h_2, the output data size of h_i and the bandwidth. Recall that we only study the case where the bandwidth of all the CUs are the same so that $t_{i,1,2}^t$ is equal to $t_{i,2,1}^t$.

2) Grouping Recursion With BPA: In this section, we extend BPA to achieve the multipartitioning and offloading of CDNNs. Specifically, the offloading targets are modified from two CUs to two CU groups (i.e., a CU group is a group of CUs), called E and C. After executing BPA, the input CDNN layers will be partitioned into two parts and offloaded to the appropriate CU groups. Next, the algorithm recursively repeats this process by executing BPA with the partitioned layers and their corresponding CU groups as input. The end sign of recursion is that there is only one CU left in the CU group. In this case, the group cannot be further divided so that the recursion should exit. When all the recursions exit, an adaptive offloading considering all the CUs' resources is completed.

Algorithm 2 GRB

Input: $\mathcal{M}$ ($\mathcal{R}$ and b), $\mathcal{G}$ (Ψ and S).

Output: The optimal multi-partitioning and offloading strategy P.

1: Initialize P;
2: **function** PARTITION($\mathcal{M}$,$\mathcal{G}$)
3: $[\mathcal{M}_e, \mathcal{M}_c] \leftarrow$ partitionM($\mathcal{M}$);
4: $[r_e, r_c, b] \leftarrow$ getParameters($\mathcal{M}_e$, $\mathcal{M}_c$);
5: $[\Psi, S] \leftarrow$ getParameters($\mathcal{G}$);
6: $[t^*, P^*] \leftarrow$ BPA(r_e, r_c, b, Ψ, S, $|\mathcal{G}|$)
7: $[\mathcal{G}_e, \mathcal{G}_c] \leftarrow$ getParts(P^*);
8: **if** $|\mathcal{M}_e| > 1$ **then**
9: PARTITION($\mathcal{M}_e$, $\mathcal{G}_e$);
10: **else**
11: Record that $\mathcal{G}_e$ should be offloaded to $\mathcal{M}_e$ in P;
12: **end if**
13: **if** $|\mathcal{M}_c| > 1$ **then**
14: PARTITION($\mathcal{M}_c$, $\mathcal{G}_c$);
15: **else**
16: Record that $\mathcal{G}_c$ should be offloaded to $\mathcal{M}_c$ in P;
17: **end if**
18: **end function**
19: return P;

The pseudocode of GRB is shown in Algorithm 2. Its input is the CU set $\mathcal{M}$ and the layer set $\mathcal{G}$. The former consists of computing power $\mathcal{R}$ and the bandwidth b, while the latter contains computation amounts Ψ and output data sizes S. GRB's output is the optimal multipartitioning and offloading strategy P. The main part of GRB is a recursive function (i.e., PARTITION), in which the input CDNN part are offloaded based on greedy and dichotomy ideas (lines 2–18). Greedy and dichotomy ideas are often simple and intuitive approaches to solving optimization problems. They provide straightforward heuristics that can be easily implemented and understood and often involve making locally optimal choices at each step, which can lead to fast convergence and reduced computational complexity. In terms of the logic of PARTITION, it first divides $\mathcal{M}$ (the input CU group) into $\mathcal{M}_e$ and $\mathcal{M}_c$ (line 3), and then it extracts r_e, r_c, b, Ψ, and S, which represent the computing power of $\mathcal{M}_e$, the computing power of $\mathcal{M}_c$, the bandwidth, the computation amounts of layers in $\mathcal{G}_e$, and the output data sizes of layers in $\mathcal{G}_e$, respectively, (lines 4 and 5). Next, BPA is executed with the extracted parameters and the number of layers $|\mathcal{G}|$ (line 6), whose output P^* is then structured to $\mathcal{G}_e$ and $\mathcal{G}_c$ (line 7). Lines 8–17 consist of two judgements. For group E, if the number of units in $\mathcal{M}_e$ is larger than one, then a new recursion is started with the input $\mathcal{M}_e$ and $\mathcal{G}_e$. Otherwise, $\mathcal{M}_e$ cannot be further divided so that P records the intermediate strategy that $\mathcal{G}_e$ should be offloaded to $\mathcal{M}_e$ and the recursion exits. When all the recursions exit, the obtained intermediate offloading strategy parts are all recorded in P. Therefore, we can partition and offload the input CDNN with the information in P.

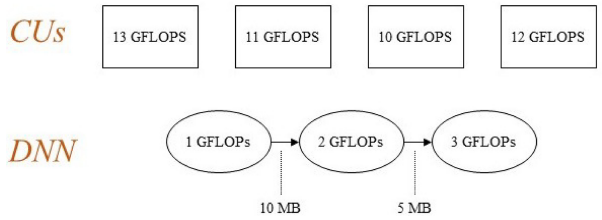


Fig. 5. Execution example of GRB (1).

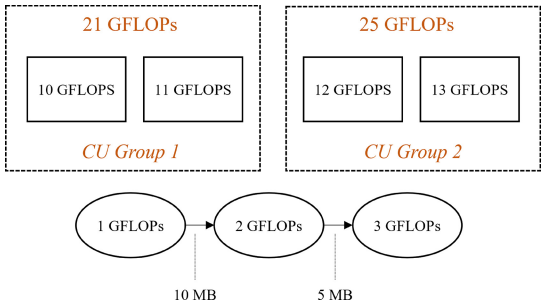


Fig. 6. Execution example of GRB (2).

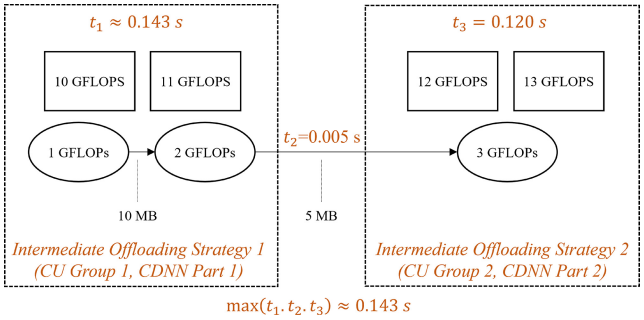


Fig. 7. Execution example of GRB (3).

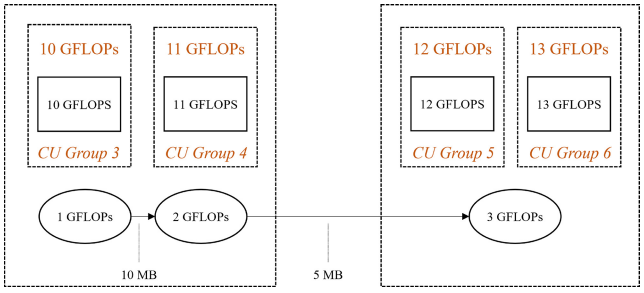


Fig. 8. Execution example of GRB (4).

C. Execution Process of GRB

In this section, we analyze the execution process of GRB with an example to help understand the algorithm. The whole partitioning and offloading process is shown in Figs. 5–9. First, Fig. 5 shows a three-layer DNN to be offloaded and four CUs participating in the offloading. The calculating amounts required for the layers are one giga floating point operations (GFLOPs), two GFLOPs, and three GFLOPs, respectively. The output sizes of the first and second layers are 10 and 5 MB, respectively. The third layer is the output layer, whose output is not considered because we ignore the return time of the final output. The computing power of the CUs are 13 giga floating-point operations per second (GFLOPS), 11 GFLOPS, ten GFLOPS, and 12 GFLOPS, respectively.

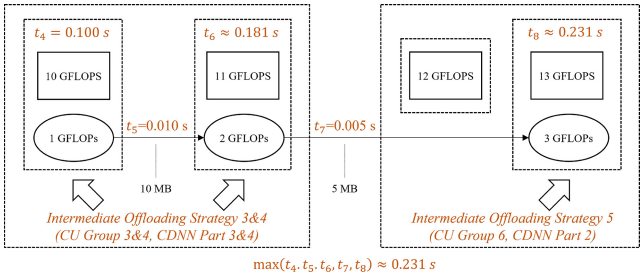


Fig. 9. Execution example of GRB (5).

GRB generates an offloading strategy based on the above parameters together with the bandwidth. We assume that the user and the CUs are in the same gigabit LAN, so the bandwidth is 1 Gb/s, denoted as B. GRB first sorts the CUs in ascending order according to their computing power, and then separate them from a specific CU. Those CUs whose computing power are less than or equal to that of this CU will be divided into a group, and those whose computing power are greater than that of this CU will be divided into another group. As shown in Fig. 6, the first CU and the second CU compose a group (CU Group 1), whose sum of computing power is 21 GFLOPS. The third and fourth CUs compose another group (CU Group 2), whose sum of computing power is 25 GFLOPS. The power difference between the two groups is four GFLOPS. The solutions where the CUs are grouped from the first, third and fourth CUs are discarded because they cause larger power differences. After grouping, GRB starts to execute BPA to partition and offload DNN into these two groups. At this time, the input parameters of BPA are the sum of computing power of CUs in the two CU groups (i.e., 21 GFLOPS and 25 GFLOPS), the bandwidth (i.e., 1 Gb/s), and the computation amounts and output sizes of the DNN layers (i.e., one GFLOPs, two GFLOPs, three GFLOPs, 10 MB, and 5 MB). BPA iterates from the first DNN layer to the last layer to obtain the optimal bi-partitioning and offloading decision with these input parameters. Denoting the two CU groups as CG_1 and CG_2, respectively, and the DNN layers as D_1, D_2, and D_3 from left to right, the intermediate offloading strategies that BPA considers can be represented as follows:

$$\{\{\mathrm{CG}_1, \{\}\}, \{\mathrm{CG}_2, \{\mathrm{D}_1, \mathrm{D}_2, \mathrm{D}_3\}\}\} \tag{15}$$

$$\{\{\mathrm{CG}_1, \{\mathrm{D}_1\}\}, \{\mathrm{CG}_2, \{\mathrm{D}_2, \mathrm{D}_3\}\}\} \tag{16}$$

$$\{\{\mathrm{CG}_1, \{\mathrm{D}_1, \mathrm{D}_2\}\}, \{\mathrm{CG}_2, \{\mathrm{D}_3\}\}\} \tag{17}$$

$$\{\{\mathrm{CG}_1, \{\mathrm{D}_1, \mathrm{D}_2, \mathrm{D}_3\}\}, \{\mathrm{CG}_2, \{\}\}\}. \tag{18}$$

Subsequently, after calculating the ATD corresponding to each strategy, BPA chooses

$$\{\{\mathrm{CG}_1, \{\mathrm{D}_1, \mathrm{D}_2\}, \{\mathrm{CG}_2, \{\mathrm{D}_3\}\}\} \tag{19}$$

as the optimal strategy with the minimum ATD. As shown in Fig. 7, the computation delay of CG_1 is $t_1 = (1+2)/21 \approx 0.143$ s, the computation delay of CG_2 is $t_2 = 3/25 = 0.120$ s,

TABLE I
COMPUTATION AMOUNTS AND DATA SIZES OF LAYERS OF VGG-16

Layer Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
computation Amount (GFLOPs)	0	0.093	1.859	0.928	1.855	0.926	1.851	1.852	0.926	1.850	1.851	0.463	0.463	0.463	0.103	0.017	0.004
Data Size (MB)	1.148	24.500	6.125	12.250	3.063	6.125	6.125	1.531	3.063	3.063	0.766	0.766	0.766	0.191	0.031	0.031	0

the transmitting delay is $t_3 = 5/1000 = 0.005$ s, and the ATD is $\max(t_1, t_2, t_3) \approx 0.143$ s.

Next, since the numbers of CUs contained in CG_1 and CG_2 are both greater than 1, GRB initiates one new recursions for each of them and repeats the above logic in the two groups. First, CG_1 is divided into CG_3 and CG_4, while CG_2 is divided into CG_5 and CG_6. The results are shown in Fig. 8. Then BPA is executed in the two recursions. The parameters are

$$\{CG_3, CG_4, D_1, D_2, B\} \tag{20}$$

and

$$\{CG_5, CG_6, D_3, B\} \tag{21}$$

respectively. The offloading strategy shown in Fig. 9 is obtained. Specifically, when offloading D_1 and D_2, BPA traverses

$$\{\{CG_3, \{\}\}, \{CG_4, \{D_1, D_2\}\}\} \tag{22}$$

$$\{\{CG_3, \{D_1\}\}, \{CG_4, \{D_2\}\}\} \tag{23}$$

and

$$\{\{CG_3, \{D_1, D_2\}, \{CG_4, \{\}\}\} \tag{24}$$

and then chooses

$$\{\{CG_3, \{D_1\}\}, \{CG_4, \{D_2\}\}\} \tag{25}$$

as the optimal solution. When offloading D_3, BPA traverses

$$\{\{CG_5, \{\}\}, \{CG_6, \{D_3\}\}\} \tag{26}$$

and

$$\{\{CG_5, \{D_3\}\}, \{CG_6, \{\}\}\} \tag{27}$$

and then chooses

$$\{\{CG_5, \{\}\}, \{CG_6, \{D_3\}\}\} \tag{28}$$

as the optimal solution. At this time, since each CG contains only one CU, no new recursion will be initiated. GRB outputs the current offloading strategy whose corresponding ATD is

$$\max\left(\frac{1}{10}, \frac{10}{1000}, \frac{2}{11}, \frac{5}{1000}, \frac{3}{13}\right) \approx 0.231 \text{ s}. \tag{29}$$

V. EXPERIMENTATION

A. Setup

DNNs: The DNNs to be partitioned and offloaded are VGG-16, NiN, AlexNet, and ResNet-18. With the input shape $3 \times 224 \times 224$, the measuring module "torchstat" of PyTorch indicates that the computation amounts (floating point operations, FLOPs) of these four DNNs are 15.5 GFLOPs (one GFLOPs $= 10^9$ FLOPs), 1.1 GFLOPs, 715.54 MFLOPs (one MFLOPs $= 10^6$ FLOPs), and 1.82 GFLOPs, respectively. It can also list the FLOPs of each DNN layer. For instance, the computation amounts of layers of VGG-16 obtained by it are shown in the second row of Table I, where the first layer is the input layer requiring no computation. We set the data size of the final (17th) layer to 0 to show that we ignore the returning delay of the final result.

Data Sizes: Besides the computation amount, another parameter corresponding to DNNs is the output data size of each layer, which is also obtained by the module torchstat. Specifically, torchstat can provide the output data shape of each layer, and then we can calculate each layer's data size with its shape and the number of bytes occupied by one datum. With the input shape $3 \times 224 \times 224$ and the byte occupation 8 (i.e., one datum occupies 8 bytes), the output of each layer is shown in the third row of Table I.

CUs: The parameters related to CUs are computing power (floating point of per second, FLOPS) and bandwidth. Thus, we combine the FLOPS values in [34] and the bandwidth values in [35] to represent CUs. For example, in the case where five units participate in the DNN offloading, their FLOPS values are five selected values from the "GFLOPS/computer" column of [34], and all their bandwidth values are one selected value from the "Maximum data rate (theoretical)" column of the "IEEE 802.11 Wi-Fi protocol summary" table in [35]. Namely, their transmitting speeds are the same.

Evaluation: We evaluate both two methods (i.e., T-GRB and T-BPA) proposed in this article, compared with DADS and the case without DNN partitioning (i.e., no majorization, NM). The metric of this experimentation is frames per second (FPS), which is the reciprocal of ATD.

Notably, DADS [15] is a milestone min-cut-based bi-partitioning method, which, however, fails to figure out a meaningful solution in some settings.. Accordingly, we supplement two modeling conditions and develop a detection-switch procedure for it to avoid its failure. Specifically, the computation delay of the first DNN layer on the cloud is set as the transmitting time of the user input (i.e., $t_1^c = t_1^t$), while the computation delay of the first DNN layer on the edge is set as infinitesimal (i.e., $t_1^e \to 0^+$). Besides, sometimes the obtained min-cut value is not equal to the ATD of the corresponding offloading strategy. Thus, we detect such meaningless solutions after running DADS and replace them with the output of T-BPA. We will specify the theoretical analysis toward DADS's failure in the future work.

B. Cooperation of Weak CUs

In this section, there are five CUs involved, whose computing power are five smallest FLOPS values in [34]. We use 54 Mb/s (802.11g), 450 Mb/s (802.11n), and 866.7 Mb/s

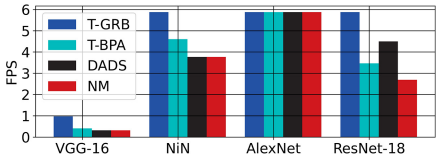


Fig. 10. Weak CU' cooperation (54 Mb/s).

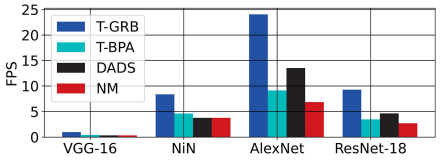


Fig. 11. Weak CU' cooperation (450 Mb/s).

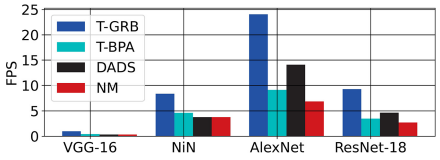


Fig. 12. Weak CU' cooperation (866.7 Mb/s).

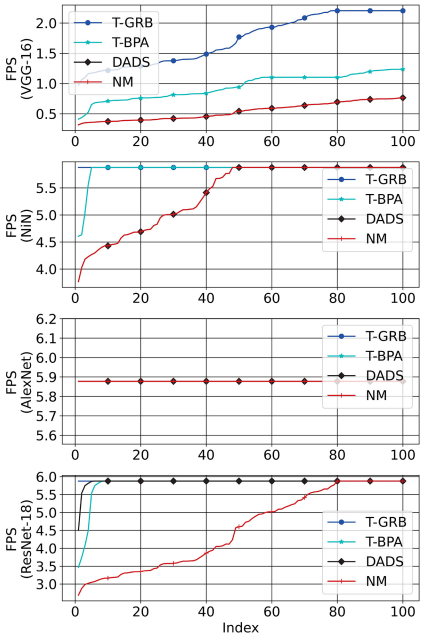


Fig. 13. Generalizability test (54 Mb/s).

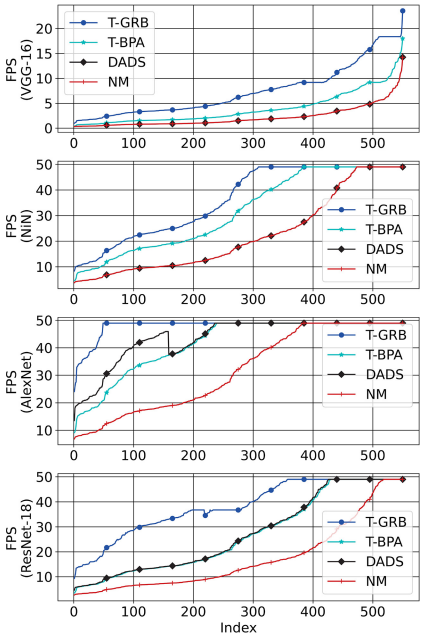


Fig. 14. Generalizability test (450 Mb/s).

(802.11ac wave1) as the bandwidth sequentially. The corresponding experimental results are shown in Figs. 10–12, respectively. With the FLOPS and bandwidth values, the scenario where five weak CUs (like smartphones and bracelets) collaboratively execute DNNs under a common WiFi network is simulated. It represents the case where DNNs are offloaded in a family or small company.

From Figs. 10–12, it can be seen that T-GRB significantly outperforms all the other methods, except the offloading of AlexNet in Fig. 10, where all the four methods show the same performance. T-BPA and DADS show the similar performance. Sometimes the former surpasses the latter, while sometimes the case is opposite. Numerically, T-GRB achieves the FPS improvement of 28%–251% compared to the others. Note that the offloading of AlexNet in Fig. 10 is not counted because it reveals that we should not partition AlexNet under such a device and network environment. Besides, the results of Figs. 11 and 12 are the same, which indicates that the bottleneck preventing FPS rising is computing power when the bandwidth reaches to 450 Mb/s. To further improve FPS, it is required to enhance units' powers.

C. Generalizability of T-GRB

In this section, there are still five CUs involved, whose computing power are five adjacent values in [34]. The bandwidth setting is the same as Section V-B. We use a window with the size 5 and step 1 to pick similar power values and run the experiments multiple times to verify the generalizability of T-GRB. Specifically, we first sort the FLOPS values in ascending order. Then, we place the window at the head of FLOPS values and move it step by step toward the tail. Each time the window is placed or moved, the four offloading methods are executed with the power values in the window. Therefore, numerous results under different combinations of CUs are obtained. We plot them as line charts are shown in Figs. 13–15, where the x-axis is the index of the experiment. It begins with 1, only indicating which experiment it is. For example, index 1 is the first experiment, where the window picks first to fifth FLOPS values as the input. The index 2 represents the second experiment, where the window moves a step so that second to sixth FLOPS values are picked. The y-axis is the FPS of offloaded VGG-16 and ResNet-18.

There are three important phenomenons reflected by Figs. 13–15.

1) T-GRB still outperforms all the others.
2) Involving more powerful units can raise the offloading performance.

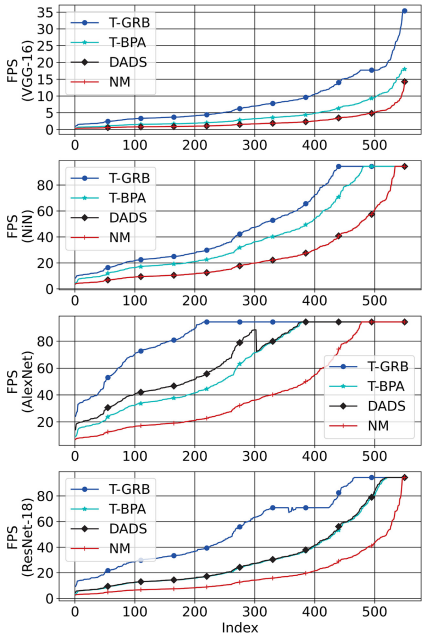


Fig. 15. Generalizability test (866.7 Mb/s).

3) Bandwidth will become the bottleneck when the involved CUs are powerful enough, and the offloading performance will stop improving.

Notably, the DADS line totally coincides with that of NM or T-BPA sometimes. The former case indicates that DADS regards the offloading strategy where all layers are offloaded to the most potent CU as optimal, while the latter case shows that DADS generates meaningless strategies so that the result of T-BPA is returned as DADS's output. Particularly, all the four lines coincide with each other in the offloading of AlexNet with 54-Mb/s bandwidth, which instructs us not to partition AlexNet under this setting.

VI. Conclusion

This article presented a more general multipartitioning and offloading method to reduce inference latency of high-precision DAG-topology DNNs for streaming tasks, which can adaptively partition DDNNs into multiple parts considering the computing power and bandwidth of all available computing units. We first derive a topological sorting-based DAG-to-chain topology transforming method and design a multipartitioning algorithm for chain-topology DNNs based on greedy and dichotomy ideas. The DDNNs' multipartitioning problem can then be solved based on the proposed transforming and multipartitioning algorithms. Experimental results show that our proposed multipartitioning method significantly outperforms bi-partitioning and nonpartitioning methods when offloading various DNN models, such as VGG-16, AlexNet, and ResNet.

References

[1] Z. Wang, Y. Zhou, Y. Shi, and W. Zhuang, "Interference management for over-the-air federated learning in multi-cell wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 8, pp. 2361–2377, Aug. 2022.

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YoLov4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[3] Z. Xu et al., "Energy-aware inference offloading for DNN-driven applications in mobile edge clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 799–814, Apr. 2021.

[4] 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.

[5] 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.

[6] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2014, pp. 4052–4056.

[7] M. Gao, W. Cui, D. Gao, R. Shen, J. Li, and Y. Zhou, "Deep neural network task partitioning and offloading for mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[8] Z. Wang, K. Liu, J. Hu, J. Ren, H. Guo, and W. Yuan, "Attrleaks on the edge: Exploiting information leakage from privacy-preserving co-inference," *Chin. J. Electron.*, vol. 32, no. 1, pp. 1–12, Jan. 2023.

[9] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[10] Y. Chang, X. Huang, Z. Shao, and Y. Yang, "An efficient distributed deep learning framework for fog-based IoT systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[11] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 3017–3030, May 2023.

[12] H. Wang, G. Cai, Z. Huang, and F. Dong, "ADDA: Adaptive distributed DNN inference acceleration in edge computing environment," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2019, pp. 438–445.

[13] G. Yoon, G.-Y. Kim, H. Yoo, S. C. Kim, and R. Kim, "Implementing practical dnn-based object detection offloading decision for maximizing detection performance of mobile edge devices," *IEEE Access*, vol. 9, pp. 140199–140211, 2021.

[14] G. Guo and J. Zhang, "Energy-efficient incremental offloading of neural network computations in mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2020, pp. 1–6.

[15] 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. (INFOCOM)*, 2019, pp. 1423–1431.

[16] S. Zhang et al., "Towards real-time cooperative deep inference over the cloud and edge end devices," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 2, pp. 1–24, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3397315

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

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

[19] 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. 2021.

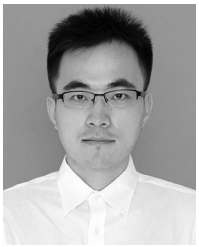
[20] Z. Chen, J. Hu, X. Chen, J. Hu, X. Zheng, and G. Min, "Computation offloading and task scheduling for DNN-based applications in cloud-edge computing," *IEEE Access*, vol. 8, pp. 115537–115547, 2020.

[21] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Mar. 2021.

[22] G. Gao, L. Wu, Z. Shao, Y. Yang, and Z. Lin, "OCDST: Offloading chained DNNs for streaming tasks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2021, pp. 1–6.

[23] Y. Yang, "Multi-tier computing networks for intelligent IoT," *Nat. Electron.*, vol. 2, no. 1, pp. 4–5, 2019.

[24] G. Ren, J. Wu, G. Li, S. Li, and M. Guizani, "Protecting intellectual property with reliable availability of learning models in AI-based cybersecurity services," *IEEE Trans. Dependable Secure Comput.*, early Access, Nov. 17, 2022, doi: 10.1109/TDSC.2022.3222972.
[25] 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.
[26] M. Zhao, X. Zhang, Z. Meng, and X. Hou, "Reliable DNN partitioning for UAV swarm," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, 2022, pp. 265–270.
[27] P. Ren, X. Qiao, Y. Huang, L. Liu, C. Pu, and S. Dustdar, "Fine-grained elastic partitioning for distributed DNN towards mobile web AR services in the 5G era," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3260–3274, Nov./Dec. 2022.
[28] J. Yu, L. Wu, G. Gao, and C. Gong, "UCL: Unit competition of layers for streaming tasks in heterogeneous networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2022, pp. 5207–5212.
[29] G. Gao, L. Wu, Y. Yang, and K. Li, "DBM: Delay-sensitive buffering mechanism for DNN offloading services," in *Proc. 27th IEEE Asia–Pacific Conf. Commun. (APCC)*, 2022, pp. 421–426.
[30] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "DNNOff: Offloading DNN-based intelligent IoT applications in mobile edge computing," *IEEE Trans. Ind. Informat.*, vol. 18, no. 4, pp. 2820–2829, Apr. 2022.
[31] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, "Service popularity-based smart resources partitioning for fog computing-enabled Industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4702–4711, Oct. 2018.
[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *Proc. 31st AAAI conf. Artif. Intell.*, 2017, pp. 4278–4284.
[33] H. Shah-Mansouri and V. W. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3246–3257, Aug. 2018.
[34] "CPU performance [EB/OL]." Rosetta@home. Accessed: Jul. 4, 2022. [Online]. Available: https://boinc.bakerlab.org/rosetta/cpu_list.php
[35] "Different Wi-Fi protocols and data rates [EB/OL]." Intel. Accessed: Jul. 4, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html



Liantao Wu (Member, IEEE) received the B.E. degree in automation from Shandong University, Jinan, China, in 2012, and the Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2017.

He is currently an Associate Professor with East China Normal University, Shanghai, China. His research interests include IoT, compressive sensing, and edge computing.

Guoliang Gao received the B.E. degree from China University of Petroleum (East China), Dongying, China, in 2019, and the M.S. degree from ShanghaiTech University, Shanghai, China, in 2022.

His research interests include DNN partitioning and edge computing.

Jing Yu received the B.E. degree from Henan Polytechnic University, Jiaozuo, China, in 2020, and the M.S. degree from ShanghaiTech University, Shanghai, China, in 2023.

Her research interests include DNN partitioning and edge computing.

Fangtong Zhou (Graduate Student Member, IEEE) received the B.E. degree from South China University of Technology, Guangzhou, China, in 2021. He is currently pursuing the M.A.Eng. degree with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China.

His research interests include federated learning and edge computing.

Yang Yang received the B.S. and M.S. degrees in radio engineering from Southeast University, Nanjing, China, in 1996 and 1999, respectively, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong, Hong Kong, in 2002.

He is currently an Associate Vice-President of Teaching and Learning, an Acting Dean of the College of Education Sciences, and a Professor with the IoT Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China. Before joining Hong Kong University of Science and Technology (Guangzhou), he has held faculty positions with The Chinese University of Hong Kong; Brunel University London, London, U.K.; University College London, London, U.K.; and Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai, China. His research interests include fog computing networks, service-oriented collaborative intelligence, wireless sensor networks, IoT applications, and advanced testbeds and experiments. He has published more than 280 papers and filed more than 80 technical patents in these research areas.

Dr. Yang has been the Chair of the Steering Committee of Asia–Pacific Conference on Communications since January 2019. In addition, he is a General Co-Chair of the IEEE DSP 2018 Conference and a TPC Vice-Chair of the IEEE ICC 2019 Conference.

Tengfei Wang (Member, IEEE) received the Ph.D. degree in mechanical engineering from Wuhan University of Technology, Wuhan, China, in 2020.

He was a Visiting Scholar with the University of California at Los Angeles, Los Angeles, CA, USA, from December 2017 to June 2019. He is currently a Lecturer with the State Key Laboratory of Maritime Technology and Safety and the School of Transportation and Logistics Engineering, Wuhan University of Technology. His research interests include multisensor perception fusion and related distributed computing methods, decision-making algorithms of intelligent ships, and other related researches.