Delay-Aware DNN Inference Throughput Maximization in Edge Computing via Jointly Exploring Partitioning and Parallelism

Jing Li†, Weifa Liang‡, Yuchen Li†, Zichuan Xu¶, and Xiaohua Jia‡
† Australian National University, Canberra, ACT 2601, Australia
‡ City University of Hong Kong, Hong Kong
¶ Dalian University of Technology, Dalian, 116020, China
Emails: jing.li5@anu.edu.au, weifa.liang@cityu.edu.hk, yuchen.li@anu.edu.au, z.xu@dlut.edu.cn, csjia@cityu.edu.hk

2021 IEEE 46th Conference on Local Computer Networks (LCN) | 978-1-6654-1886-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/LCN52139.2021.9524928

***Abstract*—Mobile Edge Computing (MEC) has emerged as a promising paradigm catering to overwhelming explosions of mobile applications, by offloading the compute-intensive tasks to an MEC network for processing. The surging of deep learning brings new vigor and vitality to shape the prospect of intelligent Internet of Things (IoT), and edge intelligence arises to provision real-time deep neural network (DNN) inference services for users. To accelerate the processing of the DNN inference of a request in an MEC network, the DNN inference model usually can be partitioned into two connected parts: one part is processed on the local IoT device of the request; and another part is processed on a cloudlet (server) in the MEC network. Also, the DNN inference can be further accelerated by allocating multiple threads of the cloudlet in which the request is assigned.**

In this paper, we study a novel delay-aware DNN inference throughput maximization problem with the aim to maximize the number of delay-aware DNN service requests admitted, by accelerating each DNN inference through jointly exploring DNN model partitioning and multi-thread parallelism of DNN inference. To this end, we first show that the problem is NP-hard. We then devise a constant approximation algorithm for it. We finally evaluate the performance of the proposed algorithm through experimental simulations. Experimental results demonstrate that the proposed algorithm is promising.

I. Introduction

The rapid development of the Internet of Things (IoT) has promoted the proliferation of numerous mobile applications including augmented reality, mobile games, and surveillance, which rely on edge devices (EDs) such as laptops, tablets, and smartphones [10]. Due to the long delays by uploading a large amount of data from IoT devices to remote clouds [11], it drives new opportunities to push deep neural network (DNN) inference to the edge of core networks by providing real-time DNN inference services [9]. Mobile Edge Computing (MEC) has become a promising paradigm to provision computing and storage resources in local cloudlets with well-trained DNN models for EDs, by offloading the whole or part of DNNs from EDs via Access Points (APs) to the MEC network for accelerating inference processing [21]. To alleviate the DNN inference delay, it is desirable that a DNN model can be partitioned into two parts: one is executed in its local IoT device and another is executed in a cloudlet of the MEC network. This task offloading method is referred to as the *DNN partitioning* method, which has been widely adopted for DNN model inferences as the amounts of output data in some intermediate layers of a DNN model are significantly smaller than that of its raw input data layer [8], thereby reducing the delay of data uploading from its IoT device to the MEC network. The potential of cloudlets on accelerating DNN inference can be further unleashed by exploring inference parallelism, i.e., leveraging multiple threads in a cloudlet to shorten the inference time further. A network provider can deploy multi-threading for DNN inference under existing frameworks (e.g. TensorFlow [2] or PyTorch [18]), or via a customized thread pool [14]. The delay-aware DNN inference service provisioning thus poses the following challenges.

For a DNN inference request with the trained DNN model and a given delay requirement, how to determine which cloudlet in the MEC network to accommodate the request? How to partition the DNN model between its local IoT device and its assigned cloudlet such that the inference delay meets the delay requirement? How many threads should be allocated for the processing of the offloaded DNN part to meet the delay requirement of the request? In this paper, we will address these challenges and devise a performance-guaranteed approximation algorithm for the delay-aware DNN inference throughput maximization problem in an edge computing environment.

The novelty of the work in this paper lies in jointly exploring DNN partitioning and inference parallelism to accelerate the DNN inference process, and an efficient approximation algorithm with a provable approximation ratio for the defined problem is devised.

The main contributions of this paper are given as follows. We first study a delay-aware DNN inference throughput maximization problem in an MEC network, with the aim to maximize the number of requests admitted, subject to computing capacities on cloudlets in the MEC network. We approach the problem by jointly partitioning the DNN model, allocating the offloaded part of the DNN to a cloudlet, and exploring inference parallelism by utilizing multiple threads in the cloudlet to meet its end-to-end inference delay requirement. We then show the NP-hardness and devise an approximation algorithm with a provable approximation ratio for the problem.

We finally conduct experimental simulations to evaluate the performance of the proposed algorithm. Experimental results demonstrate that the proposed algorithm is promising.

The rest of the paper is organized as follows. Section II reviews the related work on delay-aware DNN inference service provisioning in an edge computing environment. Section III introduces the system model, notions and notations, and defines the problems formally. The NP-hardness proof of the defined problem is given here as well. Section IV devises an approximation algorithm for the delay-aware DNN inference problem. Section V evaluates the proposed algorithm empirically, and Section VI concludes the paper.

II. Related Work

Recently, much effort focused on investigating DNN inference accelerations through task offloading in MEC environments. Mohammed *et al.* [15] devised a novel DNN partitioning scheme in an MEC network, and applied the matching theory to distribute the DNN parts to edge servers, with the aim to minimize the total computation time. Xu *et al.* [21] investigated the DNN inference offloading in an MEC network, assuming that each requested DNN has been partitioned. They then provided a randomized algorithm and an online algorithm to minimize the total energy consumption and deal with the real-time request admissions, respectively. Hu *et al.* [8] studied the DNN partitioning problem in an integrated network consisting of an edge device and a cloud. An optimal DNN partitioning strategy was proposed when the network workload is light, by transforming the problem into a minimum cut problem. They also devised an approximation approach for the problem under heavy network workload too.

The essential differences of our work in this paper from the aforementioned study [8] lie in the following aspects. (i) They focused on a single DNN model inference by minimizing the end-to-end delay on a cloud or an MEC network. In contrast, we deal with a set of DNN inference requests by admitting as many requests as possible while meeting the delay requirement of each admitted request. (ii) They assumed that the data transmission bandwidth from an IoT device of a DNN inference request is fixed, and assumed that there is only a single edge server to which the partial DNN model can be offloaded. We here assume that the data transmission rates of the IoT device of a request to different cloudlets are different, rather than fixed, and the data transmission rates are determined by both the transmission power of the IoT device and the distances of the device from cloudlets.

It must be mentioned that the work in [8] reduces the end-to-end delay minimization of a DNN inference to a minimum cut problem in an auxiliary graph. Unfortunately, the auxiliary graph constructed does not work, and thus the proposed algorithm is questionable. A brief explanation is given as follows. In their auxiliary graph construction, there is a pair of directed edges (e, v_i) and (v_i, c) that are from source e to a layer node v_i, and from the layer node v_i to the destination c with capacities $t^c_{v_i}$ and $t^e_{v_i}$ respectively, where $t^c_{v_i}$ and $t^e_{v_i}$ are the processing times of layer v_i on the cloud and edge device, respectively. Since the processing capability in the edge device is no greater than that of the cloud, the capacity of edge (e, v_i) is no greater than that of edge (v_i, c), which implies that the fractional flow from source e along edge (e, v_i) can reach the destination c through edge (v_i, c). Then all edges (e, v_i) will be saturated, and none of the uploading edges is saturated. Thus, the algorithm cannot deliver any DNN partition and the entire DNN model will be offloaded to the edge server.

There are approaches for multi-threading DNN inference acceleration, e.g., adopting framework TensorFlow [2] or PyTroch [18]. Liu *et al.* [14] also designed a customized thread pool for convolutional neural network (CNN) inference through multi-threading on multi-core central processing units (CPUs). Furthermore, the resource allocation problem under DNN inference parallelism has been investigated by researchers [16], [20]. Niu *et al.* [16] proposed a novel framework to execute a DNN model on mobile CPUs and graphics processing units (GPUs) with thread-level parallelism, by adopting a pruning-based model compression technique. Xiang *et al.* [20] included the multi-threading on multi-core CPUs with the assistance of GPUs. They presented a pipeline-based real-time DNN inference framework to deliver an efficient scheduling of CPU and GPU resources. However, their frameworks [16], [20] cannot be applied in this work, since factors such as network delay are not taken into account.

III. Preliminaries

A. System model

Given a geographical area (e.g., a metropolitan area), a set N of Access Points (APs) (or base stations) is deployed in the area. Associated with each AP, there is a co-located cloudlet (edge server) n with computing capacity C_n, which is interconnected with the AP via an optical cable and the transmission delay between them is neglected [12]. For the sake of convenience, denote by N the set of cloudlets. Since the wireless transmission range of each IoT device is fixed, the number of cloudlets the device can reach is limited, too [19]. Denote by $\mathbb{N}(r_i) \subset N$ the set of cloudlets located in the data transmission range of the IoT device of request r_i, i.e., request r_i can be served by any cloudlet $n \in \mathbb{N}(r_i)$ if the cloudlet has sufficient computing resource for processing the offloaded part of the DNN model of request r_i. Fig. 1 is an illustrative example of such a geographical area, in which the IoT device of a request can only reach a subset of cloudlets in the MEC network, i.e., any cloudlet that is within its transmission range.

In this paper, we focus on DNN inference. Compared with DNN training, the DNN inference requires much less CPU and GPU resources. Therefore, instead of making use of costly hardware accelerators (e.g., GPUs) for DNN inference, CPUs are more favored to cope with real-time DNN inference services at the network edge with cost-efficiency [17], due to their high availability and efficient scalability [7]. For simplicity, we will focus on CPU-based cloudlets, and each CPU core corresponds to a *thread*, since applying hyper-threading technique (generating two threads per CPU core) may decrease the performance due to additional context switching [14]. We

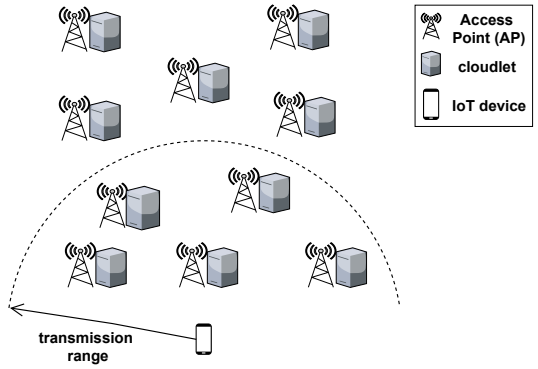


Fig. 1. An example of a geographical area with 10 APs, each of APs is co-located with a cloudlet. There is an IoT device which can offload its task to any cloudlet within its transmission range if the cloudlet has sufficient computing resource to process its offloaded task.

assume that each cloudlet $n \in N$ has a computing capacity C_n, which is measured by the number of threads (CPU cores) in it. Note that the results of this paper can be easily extended to deal with cloudlets equipped with GPU resource as well, by incorporating the threads of GPU cores and leveraging the parallelism of GPUs on inference acceleration.

B. User requests and the DNN model

There is a set R of DNN inference requests from different user IoT devices. Each request $r_i \in R$ has an end-to-end delay requirement [13], and its requested DNN can be modeled as a directed acyclic graph (DAG) $G_i = (V_i, E_i)$, where V_i is the set of inference layers and E_i is the set of layer dependencies. V_i contains $|V_i|$ inference layers: $v_{i,0}, v_{i,1}, v_{i,2}, \ldots, v_{i,|V_i-1|}$, where $v_{i,0}$ is a virtual input layer, and each of the rest inference layers can be further partitioned into sub-layers, which implies that the matrix composed by its input and output can be processed in parallel if multiple threads are allocated to the matrix multiplication computation. Each edge $(v_{i,j}, v_{i,l}) \in E_i$ represents the computation dependency of layer $v_{i,l}$ on layer $v_{i,j}$, i.e., layer $v_{i,j}$ needs to be computed first and its output is the input of layer $v_{i,l}$ with $v_{i,l} \in \mathcal{N}^+(v_{i,j})$, where $\mathcal{N}^+(v_{i,j}) = \{v_{i,l} \mid (v_{i,j}, v_{i,l}) \in E_i\}$ is the outgoing set from layer $v_{i,j}$ in G_i, Similarly, the incoming set $\mathcal{N}^-(v_{i,l})$ of $v_{i,l}$ with $\mathcal{N}^-(v_{i,l}) = \{v_{i,j} \mid (v_{i,j}, v_{i,l}) \in E_i\}$, can be defined as well. We assume that the DNN model of each request has been trained, and stored in both cloudlets and its IoT device.

Denote by $f(v_{i,j})$ the required number of floating-point operations of layer $v_{i,j}$, which is the workload to execute layer $v_{i,j}$. Note that $f(v_{i,0}) = 0$, as $v_{i,0}$ is a virtual input layer.

To utilize the processing capabilities of the local IoT device and the assigned cloudlet to accelerate DNN inference process, the layer set V_i of request r_i can be partitioned into two disjoint subsets V_i^{loc} and V_i^{mec}, i.e., $V_i = V_i^{loc} \cup V_i^{mec}$ and $V_i^{loc} \cap V_i^{mec} = \emptyset$, where V_i^{loc} and V_i^{mec} will be executed at the local IoT device and a cloudlet, respectively. Let $V_i^{tran} \subset V_i^{loc}$ be the layer set that the output of each layer $v_{i,j} \in V_i^{tran}$ will be sent to the cloudlet for processing. Note that $v_{i,0}$ is a virtual input layer which is in the DNN part on its IoT device, i.e., $v_{i,0} \in V_i^{loc}$. In case where $V_i^{mec} = V_i \setminus \{v_{i,0}\}$, and $V_i^{loc} = V_i^{tran} = \{v_{i,0}\}$, the raw input of request r_i will be transmitted to the cloudlet for DNN processing. We assume that if layer $v_{i,j} \in V_i^{mec}$ is executed in a cloudlet, then any layer $v_{i,l} \in \mathcal{N}^+(v_{i,j})$ must be executed in the cloudlet with $(v_{i,j}, v_{i,l}) \in E_i$, as the DNN is partitioned into two parts only.

C. Inference delay model

Because the result of a DNN inference usually is small, the downloading time of the inference result is negligible [21]. The end-to-end inference delay of a DNN inference request usually consists of (1) the processing delay on its local IoT device; (2) the data transmission delay from the local IoT device to its assigned cloudlet; and (3) the processing delay of the part of the DNN model in the cloudlet, which are defined as follows.

1) The processing delay on the local IoT device

Denote by $\eta_{loc}(r_i)$ the processing rate of the local IoT device of request r_i, which is measured in the number of floating-point operations per second. With $f(v_{i,j})$ the required number of floating-point operations of layer $v_{i,j}$, the processing delay $d_{i,j}^{loc}$ of layer $v_{i,j}$ on the local IoT device of request r_i then is

$$d_{i,j}^{loc} = \frac{f(v_{i,j})}{\eta_{loc}(r_i)}, \tag{1}$$

With V_i^{loc} the set of layers processed on the local IoT device, the total processing delay of the set of layers V_i^{loc} on the local IoT device of request r_i is

$$d_{loc}(V_i^{loc}) = \sum_{v_{i,j} \in V_i^{loc}} d_{i,j}^{loc}, \tag{2}$$

2) The transmission delay from the local IoT device to the assigned cloudlet

Denote by $O_{i,j}$ the output data size of layer $v_{i,j}$ in the DNN G_i of request r_i, where $O_{i,0}$ is the output of the virtual input layer $v_{i,0}$, i.e., the data size of the raw input of the DNN G_i, and the volume of $O_{i,j}$ can be obtained from the requested DNN model [21]. The data transmission rate $\lambda_{i,n}$ from the IoT device of user request r_i to cloudlet $n \in \mathbb{N}(r_i)$ is computed through the Shannon-Hartley theorem [6], i.e.,

$$\lambda_{i,n} = B_n \log_2(1 + \frac{P_i}{dist_{i,n}^{\beta} \cdot \sigma^2}) \tag{3}$$

where B_n is the bandwidth of AP co-located with cloudlet n, P_i is the transmission power of IoT device of r_i, $dist_{i,n}$ is the distance between the IoT device of r_i and AP co-located with cloudlet n [4], σ^2 is the noise power, and β is a path loss factor with $\beta = 2$ or 4 for short or long distance, respectively [4].

The transmission delay of transmitting the output of layer $v_{i,j}$ of request r_i to cloudlet $n \in \mathbb{N}(r_i)$ is

$$d_{i,j}^{tran}(n) = \frac{O_{i,j}}{\lambda_{i,n}}, \tag{4}$$

As V_i^{tran} is the set of layers whose outputs will be uploaded and transferred to cloudlet $n \in \mathbb{N}(r_i)$. Then the transmission delay of the output data by V_i^{tran} of request r_i is

$$d_{tran}(V_i^{tran}, n) = \sum_{v_{i,j} \in V_i^{tran}} d_{i,j}^{tran}(n). \tag{5}$$

3) The processing delay on a cloudlet

The inference processing rate of the offloaded layer set V_i^{mec} in a cloudlet is assumed to be proportional to the number of allocated threads (e.g., 1 thread vs 2 threads) for the offloaded part. We further assume that at most K threads can be allocated to accelerate the processing of each offloaded DNN part in a cloudlet, and all threads in any cloudlet n have the same processing rate $\eta_{mec}(n)$, which is measured by the number of floating-point operations per second. Let $k_{i,n}$ be the number of allocated threads in cloudlet $n \in \mathbb{N}(r_i)$ to process for request r_i with $1 \leq k_{i,n} \leq K$. Then, the accumulative processing rate of cloudlet n for request r_i is $k_{i,n} \cdot \eta_{mec}(n)$.

Recall that $f(v_{i,j})$ is the required number of floating-point operations of layer $v_{i,j}$, the processing delay of layer $v_{i,j}$ in cloudlet n with $k_{i,n}$ allocated threads for request r_i is

$$d_{i,j}^{mec}(n, k_{i,n}) = \frac{f(v_{i,j})}{k_{i,n} \cdot \eta_{mec}(n)}, \tag{6}$$

The processing delay of the offloaded layer set V_i^{mec} of request r_i in cloudlet n with $k_{i,n}$ allocated threads thus is

$$d_{mec}(V_i^{mec}, n, k_{i,n}) = \sum_{v_{i,j} \in V_i^{mec}} d_{i,j}^{mec}(n, k_{i,n}), \tag{7}$$

4) The end-to-end delay

The end-to-end delay of offloading the layer set V_i^{mec} to cloudlet $n \in \mathbb{N}(r_i)$ for request r_i is

$$\begin{aligned} d_{d2d}(V_i^{loc}, V_i^{mec}, n, k_{i,n}) =& d_{loc}(V_i^{loc}) + d_{tran}(V_i^{tran}, n) \\ &+ d_{mec}(V_i^{mec}, n, k_{i,n}). \end{aligned} \tag{8}$$

To admit request r_i by meeting its delay requirement D_i, we must have

$$d_{d2d}(V_i^{loc}, V_i^{mec}, n, k_{i,n}) \leq D_i. \tag{9}$$

D. Problem definition

Definition 1: Given a set N of cloudlets co-located with $|N|$ APs in a geographical area, and a set of requests R issued from IoT devices, each DNN inference request $r_i \in R$ issued from an IoT device has a DNN inference model G_i with the end-to-end inference delay requirement D_i, assuming that the model has been stored in both cloudlets and the local IoT device already. To accelerate the DNN inference, each DNN model is partitioned into two parts: one part is executed in its local IoT device, and another part is offloaded to a cloudlet within the transmission range of the IoT device, and is allocated with up to K (≥ 1) threads in the cloudlet for processing, the *delay-aware DNN inference throughput maximization problem* in such an MEC environment is to maximize as many DNN inference requests admitted as possible while meeting their end-to-end-delay requirements, subject to computing capacity on each cloudlet in N.

Theorem 1: The delay-aware DNN inference throughput maximization problem is NP-hard.

The detailed proof is omitted, due to space limitation.

IV. Approximation algorithm for the delay-aware DNN inference throughput maximization problem

In this section, we deal with the delay-aware DNN inference throughput maximization problem by devising an approximate solution for it. Specifically, we first consider offloading part of the DNN model of request r_i to cloudlet $n \in \mathbb{N}(r_i)$ with $k_{i,n}$ allocated threads, $1 \leq k_{i,n} \leq K$. If such an assignment is feasible (i.e., meeting its end-to-end delay requirement), cloudlet n will become a candidate for the assignment of r_i. We then determine the minimum number $k_{i,n}^{min}$ of threads needed to meet the delay requirement. We finally reduce the problem to a maximum profit generalized assignment problem (GAP). Thus, an approximate solution to the latter in turn returns an approximate solution to the former. We also analyze the correctness and time complexity of the proposed approximation algorithm.

A. Partitioning the DNN model and offloading part of it to cloudlet n with $k_{i,n}$ allocated threads

We assume that part of the DNN model of request r_i will be offloaded to cloudlet n with $k_{i,n}$ allocated threads for processing, $1 \leq k_{i,n} \leq K$. To this end, we reduce this partitioning problem into the maximum flow and minimum cut problem in an auxiliary graph $G'_{i,n,k_{i,n}}$. We then show that the minimum cut in $G'_{i,n,k_{i,n}}$ corresponds to an optimal partition of the DNN, and the value of the minimum cut is the minimum end-to-end delay by offloading part of the DNN model to cloudlet n with $k_{i,n}$ allocated threads. If this minimum end-to-end delay meets the delay requirement D_i, the partition of the DNN or part of DNN offloading is feasible.

In the following, we construct the auxiliary graph $G'_{i,n,k_{i,n}}$ for the DNN model partition of request r_i to cloudlet n with $k_{i,n}$ allocated threads, where $n \in \mathbb{N}(r_i)$ and $1 \leq k_{i,n} \leq K$. Recall that $d_{i,j}^{tran}(n)$ represents the transmission delay of the output of layer $v_{i,j}$ to cloudlet n, and $d_{i,j}^{mec}(n, k_{i,n})$ represents the processing delay of layer $v_{i,j}$ on cloudlet n with $k_{i,n}$ threads, respectively. For the sake of convenience, in the rest of discussions, we substitute the notations of $d_{i,j}^{tran}(n)$ and $d_{i,j}^{mec}(n, k_{i,n})$, with $d_{i,j}^{tran}$ and $d_{i,j}^{mec}$, respectively.

Given a DNN model $G_i = (V_i, E_i)$, the construction of the auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$ with edge capacity $w(\cdot)$ is given as follows, where $V'_i = \{v_{i,j}, v'_{i,j} \mid v_{i,j} \in V_i\} \cup \{s, t\}$ and $E'_i = \{(s, v_i), (v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i \backslash \{v_{i,0}\}\} \cup \{(v'_{i,j}, v_{i,l}), (v_{i,l}, v_{i,j}) \mid (v_{i,j}, v_{i,l}) \in E_i\}$. Specifically, we add two virtual nodes s and t as the source and sink nodes, respectively. We add two nodes $v_{i,j}$ and $v'_{i,j}$ for each node $v_{i,j} \in V_i$. We also add edges $(s, v_{i,j})$ and $(v_{i,j}, v'_{i,j})$ for each node $v_{i,j} \in V_i$, together with the edges $(v_{i,j}, t)$ for each node $v_{i,j} \in V_i \setminus \{v_{i,0}\}$. We also add edges $(v'_{i,j}, v_{i,l})$ and $(v_{i,l}, v_{i,j})$ for each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in G_i.

The edge capacity assignment of $G'_{i,n,k_{i,n}}$ as well as the assignment justification is given as follows.

The capacity $w(s, v_{i,0})$ of edge $(s, v_{i,0}) \in E'_i$ is infinite, which implies that the raw input data of request r_i is at its local IoT device initially, i.e., $w(s, v_{i,0}) = \infty$.

For each edge $(s, v_{i,j}) \in E'_i$ with $v_{i,j} \in V_i \setminus \{v_{i,0}\}$, its capacity is the processing delay $d^{mec}_{i,j}$ of layer $v_{i,j}$ on cloudlet n with $k_{i,n}$ allocated threads, i.e., $w(s, v_{i,j}) = d^{mec}_{i,j}$.

For each edge $(v_{i,j}, t) \in E'_i$ with $v_{i,j} \in V_i \setminus \{v_{i,0}\}$, its capacity is the processing delay $d^{loc}_{i,j}$ of layer $v_{i,j}$ on the local IoT device of request r_i, i.e., $w(v_i, t) = d^{loc}_{i,j}$. Note that $v_{i,0}$ is a virtual input layer on the local IoT device, therefore, edge $(v_{i,0}, t)$ is not included in the auxiliary graph.

For each edge $(v_{i,j}, v'_{i,j}) \in E'_i$ with $v_{i,j} \in V_i$, its capacity is the transmission delay of the output of layer $v_{i,j}$ to cloudlet n, i.e., $w(v_{i,j}, v'_{i,j}) = d^{tran}_{i,j}$.

For each edge $(v'_{i,j}, v_{i,l}) \in E'_i$ with $(v_{i,j}, v_{i,l}) \in E_i$, its capacity is set as infinity, i.e., $w(v'_{i,j}, v_{i,l}) = \infty$. This is because the transmission delay $d^{tran}_{i,j}$ of the output of $v_{i,j}$ has been assigned to the edge $(v_{i,j}, v'_{i,j})$. If the output of $v_{i,j}$ is transmitted to the cloudlet to process all its successor layers $\mathcal{N}^+(v_{i,j})$, $d^{tran}_{i,j}$ is counted only once by including edge $(v_{i,j}, v'_{i,j})$ in a minimum cut of the auxiliary graph,

For each edge $(v_{i,l}, v_{i,j}) \in E'_i$ with $(v_{i,j}, v_{i,l}) \in E_i$, its capacity is set as infinity, i.e., $w(v_{i,l}, v_{i,j}) = \infty$. This is due to the following reason. Given a potential cut M (set of edges) on the auxiliary graph, V'_i is partitioned into two sets V_s and V_t, i.e., $V_s \cup V_t = V'_i$ and $V_s \cap V_t = \emptyset$, where source $s \in V_s$ and sink $t \in V_t$. For each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$, we have that if $v_{i,j} \in V_t$, then $v_{i,l} \in V_t$, which implies that if a layer $v_{i,j}$ is executed on a cloudlet, its successor layer $v_{i,l}$ has to be executed on the cloudlet too. Similarly, we have that if $v_{i,l} \in V_s$, then $v_{i,j} \in V_s$, for each $(v_{i,j}, v_{i,l}) \in E_i$. The claims will be shown in Lemma 2.

One example of the construction of the auxiliary flow network is illustrated in Fig. 2. A potential cut in it is $M = \{(v_{i,1}, t), (v_{i,1}, v'_{i,1}), (s, v_{i,2}), (s, v_{i,3}), (s, v_{i,4})\}$, and the set V_i is partitioned into two disjoint sets V_s and V_t, where $V_s = \{s, v_{i,0}, v'_{i,0}, v_{i,1}\}$ and $V_t = \{t, v'_{i,1}, v_{i,2}, v'_{i,2}, v_{i,3}, v'_{i,3}, v_{i,4}, v'_{i,4}\}$. This implies that $v_{i,1}$ is executed in the local IoT device, i.e., $V^{loc}_i = V_s \cap V_i = \{v_{i,0}, v_{i,1}\}$, where $v_{i,0}$ is the virtual input layer. While $v_{i,2}, v_{i,3}$, and $v_{i,4}$ are executed in cloudlet $n \in \mathbb{N}(r_i)$ with $k_{i,n}$ allocated threads, i.e., $V^{mec}_i = V_t \cap V_i = \{v_{i,2}, v_{i,3}, v_{i,4}\}$.

It can be seen that the minimum cut M^* in $G'_{i,n,k_{i,n}}$ corresponds to a two-partitioning $\{V^{loc}_i, V^{mec}_i\}$ of the DNN G_i that minimizes the end-to-end delay of request r_i when r_i is assigned to cloudlet n with $k_{i,n}$ threads, where $V^{loc}_i = V_s \cap V_i$ and $V^{mec}_i = V_t \cap V_i$. In other words, $\sum_{e \in M^*} w(e)$ is the entire inference delay of the DNN by the partitioning of $\{V^{loc}_i, V^{mec}_i\}$. This claim will be shown rigorously later.

B. Approximation algorithm

Having partitioned the DNN model of each request into two connected components, the delay-aware DNN inference throughput maximization problem can be solved through a reduction to a maximum profit GAP as follows.

If the DNN part of request r_i can be offloaded to cloudlet $n \in \mathbb{N}(r_i)$ with $k_{i,n}$ allocated threads to meet its delay requirement D_i, then a minimum number $k^{min}_{i,n}$ of allocated threads in cloudlet n for request r_i with $1 \leq k^{min}_{i,n} \leq k_{i,n}$ needs to be identified while still meeting the delay requirement. This can be achieved by binary search on the value range $[1, K]$ of $k^{min}_{i,n}$ by constructing no more than $\lceil \log K \rceil$ auxiliary graphs.

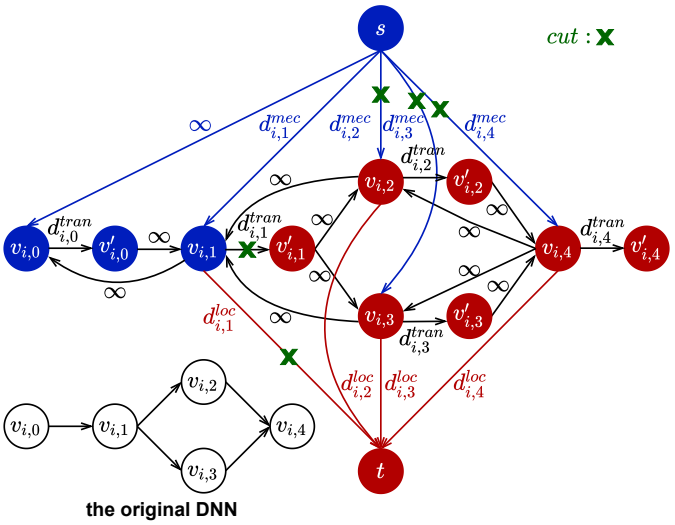


Fig. 2. An illustrative example of the auxiliary flow network $G'_{i,n,k_{i,n}}$ driven from a DNN (consisting of 4 layers) of request r_i, by assigning part of the DNN to cloudlet n with $k_{i,n}$ allocated threads for the DNN inference process.

Denote by $p(i, n)$ the throughput gain if request r_i is assigned to cloudlet n, where $p(i, n) = 0$ indicates that either (i) cloudlet n is not within the transmission range of the IoT device of request r_i, i.e., $n \notin \mathbb{N}(r_i)$; or (ii) request r_i is not admissible by assigning it to cloudlet n with up to K allocated threads, i.e., its delay requirement cannot be met with any DNN partitioning strategy, even if request r_i is assigned to cloudlet n by allocating it with the maximum number of threads K; and $p(i, n) = 1$ otherwise. Note that in case $p(i, n) = 0$, the value of $k^{min}_{i,n}$ is set as ∞.

The problem reduction proceeds as follows. For each cloudlet $n \in \mathcal{N}$, there is a corresponding bin b_n with capacity of C_n. While for each request r_i, there is a corresponding item i with $1 \leq i \leq |R|$. If item i can be assigned to bin b_n with $k^{min}_{i,n}$ allocated threads while meeting its delay requirement, it has a profit of $p(i, n)$ $(= 1)$ with a size of $k^{min}_{i,n}$; otherwise, its profit is $p(i, n) = 0$ with a size of ∞, the problem then to pack as many items as possible to the $|N|$ bins such that the accumulative profit is maximized, subject to the capacity on each bin. This is the maximum profit GAP, and there is an approximation algorithm for it in [5].

The detailed algorithm for the delay-aware DNN inference problem is then given in `Algorithm 1`.

C. Algorithm analysis

In the following, we analyze the properties of the constructed auxiliary graph $G'_{i,n,k_{i,n}}$, and show the correctness of the proposed approximation algorithm. We also analyze the approximation ratio and time complexity of the algorithm.

Lemma 1: Given a DNN $G_i = (V_i, E_i)$, its layer set V_i is partitioned into two disjoint subsets V^{loc}_i and V^{mec}_i, where V^{loc}_i and V^{mec}_i are executed in the local IoT device and a cloudlet in edge computing, respectively. For each layer

Algorithm 1 An approximation algorithm for the delay-aware DNN inference problem

Input: A set N of cloudlets co-located with APs in a monitoring area, a set of requests R, and a given integer $K \geq 1$.
Output: Maximize the network throughput by admitting as many DNN inference requests as possible.

for each request $r_i \in R$ **do**
  for each cloudlet $n \notin \mathbb{N}(r_i)$ **do**
    $k_{i,n}^{min} \leftarrow \infty$; $p(i,n) \leftarrow 0$;
  end for
  for each cloudlet $n \in \mathbb{N}(r_i)$ **do**
    Construct auxiliary graph $G'_{i,n,K}$;
    Calculate the end-to-end delay $d_{d2d}(r_i, n, K)$ by finding a minimum cut in $G'_{i,n,K}$;
    if $d_{d2d}(r_i, n, K) > D_i$ **then**
      $k_{i,n}^{min} \leftarrow \infty$; $p(i,n) \leftarrow 0$; /* the delay requirement of request r_i cannot be met when assigned to cloudlet n with K threads.*/
    else
      $k_l \leftarrow 1$; $k_r \leftarrow K$; /* Use binary search to find a minimum number $k_{i,n}^{min}$ of threads to meet the delay requirement D_i of request r_i when it is assigned to cloudlet n/*
      while $k_l \leq k_r$ **do**
        $k_m \leftarrow \lfloor (k_l + k_r)/2 \rfloor$;
        Construct auxiliary graph G'_{i,n,k_m};
        Calculate the end-to-end delay $d_{d2d}(r_i, n, k_m)$ by finding a minimum cut in G'_{i,n,k_m};
        if $d_{d2d}(r_i, n, k_m) \leq D_i$ **then**
          $k_r \leftarrow k_m - 1$;
        else
          $k_l \leftarrow k_m + 1$;
        end if
      end while
      $k_{i,n}^{min} \leftarrow k_l$; $p(i,n) \leftarrow 1$;
    end if
  end for
end for
Construct a maximum profit GAP instance, where each cloudlet n corresponds to a bin b_n with a capacity C_n, each request r_i corresponds to an item i with a size $k_{i,n}^{min}$ and a profit $p(i,n)$ if assigned to bin b_n;
An approximate solution $\mathbb{A}$ is obtained, by invoking the approximation algorithm for the maximum profit GAP in [5];
return the solution $\mathbb{A}$ to the delay-aware DNN inference problem.

dependency $(v_{i,j}, v_{i,l}) \in E_i$ in the DNN G_i, we have (i) if $v_{i,j} \in V_i^{mec}$, then $v_{i,l} \in V_i^{mec}$; (ii) if $v_{i,l} \in V_i^{loc}$, then $v_{i,j} \in V_i^{loc}$; and (iii) $v_{i,0} \in V_i^{loc}$.

This is because we assume that the DNN model G_i can only be partitioned into two parts.

Lemma 2: Given a DNN inference model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, let V_s and V_t be the sets of nodes in $G'_{i,n,k_{i,n}}$ partitioned by a potential cut M, where V_s contains source node s and V_t contains destination node t, respectively. For each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in G_i, we have (i) if $v_{i,j} \in V_t$, then $v_{i,l} \in V_t$; (ii) if $v_{i,l} \in V_s$, then $v_{i,j} \in V_s$; and (iii) $v_{i,0} \in V_s$.

Proof In the construction of $G'_{i,n,k_{i,n}}$, for each layer dependency $(v_{i,j}, v_{i,l}) \in E_i$ in G_i, an edge $(v_{i,l}, v_{i,j})$ with capacity of infinity is added to $G'_{i,n,k_{i,n}}$. This implies that edge $(v_{i,l}, v_{i,j})$ in G'_i cannot be included in M, and $v_{i,j}$ always is reachable from $v_{i,l}$. We now show claim (i) by contradiction, i.e., we assume that $v_{i,l} \in V_s$. However, $v_{i,j} \in V_t$ and $v_{i,l}$ can reach $v_{i,j}$ through edge $(v_{i,l}, v_{i,j})$. This contradicts the definition of a cut. The proof of claim (ii) is similar to that of claim (i), omitted. Claim (iii) always holds, because the capacity of edge $(s, v_{i,0})$ is infinity and node s can reach $v_{i,0}$.

Lemma 3: Given a DNN model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, let V_s and V_t be the sets of nodes in $G'_{i,n,k_{i,n}}$ partitioned by a minimum cut, where V_s contains s and V_t contains t, respectively. Let M^* be the set of edges in the minimum cut. In the auxiliary graph $G'_{i,n,k_{i,n}}$, (i) if $v_{i,j} \in V_t \cap V_i$ with $v_{i,j} \in V_i \setminus \{v_0\}$, then edge $(s, v_{i,j}) \in M^*$ and edge $(v_{i,j}, t) \notin M^*$; (ii) if $v_{i,j} \in V_s \cap V_i$ with $v_{i,j} \in V_i \setminus \{v_0\}$, then edge $(v_{i,j}, t) \in M^*$ and edge $(s, v_{i,j}) \notin M^*$; and (iii) edge $(s, v_{i,0}) \notin M^*$.

Proof (i) Since edge $(s, v_{i,j})$ directly connects source s to $v_{i,j}$, edge $(s, v_{i,j})$ must be included in M^* by the definition of the minimum cut. Assuming that $(v_{i,j}, t) \in M^*$, as $v_{i,j} \in V_t$, it can be seen that the removal of $(v_{i,j}, t)$ from M^* results in a new cut M' with a smaller value, i.e., $M^* = M' \cup \{(v_{i,j}, t)\}$ and $\sum_{e \in M'} w(e) < \sum_{e \in M^*} w(e)$. This contradicts the assumption that M^* is a minimum cut. Thus, $(v_{i,j}, t) \notin M^*$. (ii) The proof is similar to (i), omitted. (iii) This is due to the fact that the capacity of edge $(s, v_{i,0})$ is infinite.

Lemma 4: Given a DNN model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$. Each potential DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ in G_i corresponds to a potential cut M in $G'_{i,n,k_{i,n}}$.

Proof Similar to Lemma 3, we construct such a cut M on $G'_{i,n,k_{i,n}}$ by any DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$, where $M = \{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \cup \{(v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i^{tran}\}$. Then we have $V_s = \{s\} \cup \{v_{i,j} \mid v_{i,j} \in V_i^{tran}\} \cup \{v_{i,j}, v'_{i,j} \mid v_{i,j} \in V_i^{loc} \setminus V_i^{tran}\}$, and $V_t = \{t\} \cup \{v'_{i,j} \mid v_{i,j} \in V_i^{tran}\} \cup \{v_{i,j}, v'_{i,j} \mid v_{i,j} \in V_i^{mec}\}$.

We now show that M is a feasible cut in $G_{i,n,k_{i,n}}$, i.e., we show that for the given cut M, (i) any node in V_s is reachable from s; and (ii) any node in V_t is not reachable from s.

(i) Source s can reach any node $v_{i,j} \in V_i^{loc}$, since edge $(s, v_{i,j})$ with $v_{i,j} \in V_i^{loc}$ is not added to cut M. Source s can also reach any node $v'_{i,j}$ with $v_{i,j} \in V_i^{loc} \setminus V_i^{tran}$ through edges $(s, v_{i,j})$ and $(v_{i,j}, v'_{i,j})$, because edge $(v_{i,j}, v'_{i,j})$ with $v_{i,j} \in V_i^{loc} \setminus V_i^{tran}$ is not added to cut M.

(ii) We have $\{v'_{i,j} \mid v_{i,j} \in V_i^{tran}\} \subset V_t$, because for each $v_{i,j} \in V_i^{tran}$, node $v'_{i,j}$ has only one incoming edge $(v_{i,j}, v'_{i,j})$, however, $(v_{i,j}, v'_{i,j}) \in M$, then $v'_{i,j}$ is not reachable from s. We now show that $\{v_{i,j} \mid v_{i,j} \in V_i^{mec}\} \subset V_t$ by a contradiction. Assume that there exists a node $v_{i,j} \in V_i^{mec}$, which is reachable from source s. Because $\{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \subset M$, source s can reach $v_{i,j}$ only if source s first reaches a node $v_{i,a} \in V_{loc}$ with edge $(s, v_{i,a})$. Also, the path from any node $v_{i,a} \in V_{loc}$ to any node $v_{i,j} \in V_{mec}$ in $G'_{i,n,k_{i,n}}$ must pass through an edge in $\{(v_{i,b}, v'_{i,b}) \mid v_{i,b} \in V_i^{tran}\}$ by the construction of the auxiliary graph. However, with $\{(v_{i,b}, v'_{i,b}) \mid v_{i,b} \in V_i^{tran}\} \subset M$, source s cannot reach any node $v_{i,j} \in V_i^{mec}$ though any node $v_{i,a} \in V_{loc}$, which results in a contradiction. We have $\{v'_{i,j} \mid v_{i,j} \in V_i^{mec}\} \subset V_t$, because each $v'_{i,j}$ has only one incoming edge $(v_{i,j}, v'_{i,j})$ with $v_{i,j} \in V_i^{mec}$, but s cannot reach any node $v_{i,j} \in V_i^{mec}$ as mentioned. To reach sink t, source s must first reach any node in V_i^{mec}, due to $\{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \subset M$.

However, source s cannot reach any node $v_{i,j} \in V_i^{mec}$ as mentioned. Therefore, source s cannot reach sink t and $t \in V_t$.

Lemma 5: Given a DNN $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, let V_s and V_t be the node sets of $G'_{i,n,k_{i,n}}$ partitioned by a minimum cut M^*, where V_s contains s and V_t contains t. The minimum cut M^* in $G'_{i,n,k_{i,n}}$ corresponds to a feasible DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ of G_i, where $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$.

Proof Let $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$, such a DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ is feasible, because the minimum cut in $G'_{i,n,k_{i,n}}$ and a feasible DNN partitioning in G_i have the same patterns by Lemma 1 and 2. Especially, we have $\{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \subset M^*$, by Lemma 3. We can also obtain the set of layers $V_i^{tran} \subset V_i^{loc}$, the successor layers of which are in V_i^{mec}. We then show $\{(v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i^{tran}\} \subset M^*$ by a contradiction. Assume that it exists a node $v_{i,j} \in V_{tran}$ with $(v_{i,j}, v'_{i,j}) \notin M^*$, and there is a layer dependency $(v_{i,j}, v_{i,l})$ with $v_{i,j} \in V_{tran} \subset V_i^{loc}$ and $v_{i,l} \in V_{mec}$. As $V_i^{loc} = V_s \cap V_i$ and $V_i^{mec} = V_t \cap V_i$, we have $v_{i,j} \in V_s$ and $v_{i,l} \in V_t$. However, the capacity of edge $(v'_{i,j}, v_{i,l})$ is infinite, and $v_{i,j}$ can reach $v_{i,l}$ though edges $(v_{i,j}, v'_{i,j})$ and $(v'_{i,j}, v_{i,l})$, which contradicts the definition of a cut. Thus we have the minimum cut $M^* = \{(s, v_{i,j}) \mid v_{i,j} \in V_i^{mec}\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i^{loc} \setminus \{v_{i,0}\}\} \cup \{(v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i^{tran}\}$, because M^* is already a cut of $G'_{i,n,k_{i,n}}$, which has been shown in Lemma 4.

Lemma 6: Given a DNN model $G_i = (V_i, E_i)$ and its auxiliary flow network $G'_{i,n,k_{i,n}} = (V'_i, E'_i)$, the minimum cut M^* of $G'_{i,n,k_{i,n}}$ corresponds to a feasible DNN partitioning with the minimum end-to-end delay.

Lemma 6 can be derived from Lemma 4 and Lemma 5.

Theorem 2: Given a set N of cloudlets co-located with APs in a monitoring area, and a set R of DNN inference requests with delay requirements, there is a $\frac{1}{2+\epsilon}-$approximation algorithm, `Algorithm 1`, for the delay-aware DNN inference throughput maximization problem, which takes $O(|R| \cdot |N| \cdot \lceil \log K \rceil \cdot (|V|_{max} + |E|_{max}) \cdot |V|^2_{max} + |R| \cdot |N| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$ time, where ϵ is a constant with $0 < \epsilon \leq 1$, K is the maximum number of threads allocated to a request in any cloudlet, $|V|_{max}$ and $|E|_{max}$ are the maximum numbers of layers and edges in a DNN of any request, respectively.

Proof Given the assigned cloudlet n and the number of allocated threads, a minimum cut in the constructed auxiliary graph results in an optimal DNN partitioning to minimize the end-to-end delay of a request r_i by Lemma 6, and a minimum number of allocated threads for request r_i to meet its delay requirement is then found if request r_i is assigned to cloudlet n. The approximation ratio of `Algorithm 1` is $\frac{1}{2+\epsilon}$, derived from the approximation algorithm in [5] directly.

The time complexity of `Algorithm 1` is analyzed as follows. There are at most $|R| \cdot |N| \cdot \lceil \log K \rceil$ auxiliary graphs for $|R|$ requests to be constructed, while it takes $O((|V|_{max} + |E|_{max}) \cdot |V|^2_{max})$ time to find a minimum cut in each auxiliary graph by the algorithm in [3]. Also, it takes $O(|R| \cdot |N| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$ time for the maximum GAP, by the approximation algorithm in [5].

V. Performance Evaluation

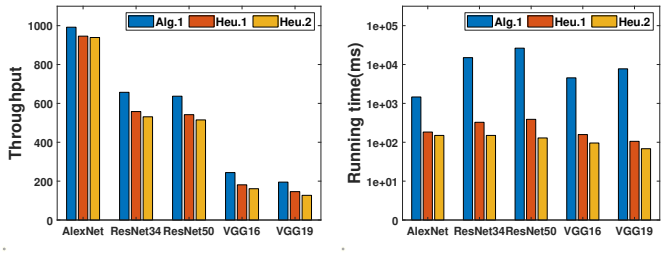
A. Experimental Settings

The geographical area is a 1 km × 1 km square area [15] and there are 100 cloudlets deployed on a regular 10×10 grid MEC network, each of which is co-located with an AP. The bandwidth of each AP varies from 2 MHz to 10 MHz [8], and each AP can provide services to a user within 100 m [19]. There are $1,000$ DNN inference requests, each of which is issued from an IoT device, and the IoT devices are randomly scattered over such a geographical area. For DNN inference, we consider the well-known DNNs: { AlexNet, ResNet34, ResNet50, VGG16, VGG19 }. Each request contains an image for inference, which is extracted from the videos in the self-driving dataset BDD100K [22]. The transmission power of each IoT device is randomly drawn between 0.1 Watt and 0.5 Watt [4], [19], noise power σ^2 is set as 1×10^{-10} Watt, and the path loss factor β is set as 4 [4]. The delay requirement of a request varies from 0.1 s to 0.3 s. Each cloudlet is assumed to be equipped with 32, 48, or 64 CPU cores [1]. We further assume that at most 10 threads (CPU cores) can be allocated for a request in any cloudlet, i.e., $K = 10$. All CPU cores in a cloudlet are assumed to share the same clock speed, which varies from 2.5 GHz to 3 GHz in different cloudlets [1]. And the clock speed of each IoT device ranges from 0.5 GHz to 1 GHz [4]. Each cloudlet or IoT device is assumed to conduct 4 floating-point operations per cycle [15]. The parameter ϵ in `Algorithm 1` is set as 0.5 by Theorem 2. The value in each figure is the mean of the results out of 15 different runs of MEC network instances of the same size. The running time of each algorithm is obtained, based on a desktop with a 3.60 GHz Intel 8-Core i7-7700 CPU and 16 GB RAM. Unless specified, the above parameters are adopted by default.

To evaluate the performance of `Algorithm 1` (referred to as `Alg.1`) for the delay-aware DNN inference problem, we here introduce two heuristic algorithms as benchmarks: algorithms `Heu.1` and `Heu.2`. Algorithm `Heu.1` is based on an existing DNN partitioning strategy *Neurosurgeon* [9], which, however, only works for chain-topology DNNs. Therefore, we preprocess each DAG-topology DNN by a topological sorting approach as did in [8], and then adopt Neurosurgeon to partition the DNN between the local IoT device and a cloudlet. For each request r_i, we now can find the minimum number $k_{i,n}^{min}$ of needed threads among all cloudlets to meet its delay requirement. Finally, algorithm `Heu.1` admits a request with the minimum number $k_{i,n}^{min}$ of threads among all requests, iteratively, until no more request can be admitted. Algorithm `Heu.2` offloads the entire DNN of a request to its nearest cloudlet with sufficient available threads to meet its delay requirement. Similarly, algorithm `Heu.2` admits a request with the minimum number of needed threads among all requests, iteratively, until no more requests can be admitted.

B. Performance evaluation of the proposed algorithm for the delay-aware DNN inference throughput maximization problem

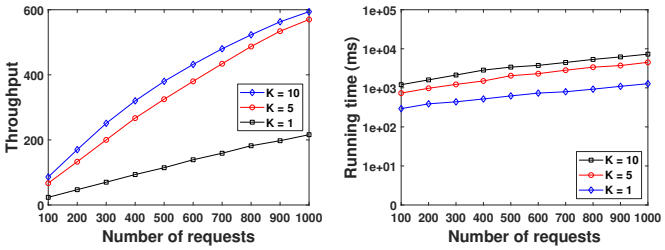
We first evaluated the performance of algorithm `Alg.1` against algorithms `Heu.1` and `Heu.2` with $1,000$ requests, over the considered DNNs, respectively. Fig. 3 demonstrates the throughput and running time delivered by different algorithms. It can be seen from Fig. 3(a) that algorithm `Alg.1` admits more requests than algorithms `Heu.1` and `Heu.2` in all cases. For example, for ResNet34, algorithm `Alg.1` outperforms algorithms `Heu.1` and `Heu.2` by 17.7% and 23.7%, respectively. This is because algorithm `Alg.1` establishes an efficient DNN partitioning strategy for each inference request, and makes resource-efficient decisions of request admissions. Also, the performance of algorithm `Alg.1` over VGG19 is 19.7% of itself over AlexNet. This is because the inference over VGG19 requires the largest number of floating-point operations (about 19.6 G), while the inference over AlexNet requires the smallest number of floating-point operations (about 0.7 G).



(a) The throughput (b) The running time

Fig. 3. Performance of different algorithms for the delay-aware DNN inference throughput maximization problem over different DNNs.

We then studied the impact of parameter K on the performance of algorithm `Alg.1`, by varying the number of requests from 100 to $1,000$, where the requested DNN of each request is randomly drawn from the predefined DNN set. Fig. 4 illustrates the throughput and running time of algorithm `Alg.1` when $K = 1$, 5, and 10, respectively. From Fig. 4(a), we can see that when the number of requests is $1,000$, the performance of algorithm `Alg.1` when $K = 1$ is 36.4% of itself when $K = 10$. This is due to the fact that a larger value of K implies that more threads can be allocated to accelerate the DNN inference of a request to meet its delay requirement.



(a) The throughput (b) The running time

Fig. 4. Impact of parameter K on the performance of algorithm `Alg.1`.

VI. Conclusion

In this paper, we studied a delay-aware DNN inference throughput maximization problem to maximize the number of requests admitted, subject to the computing capacities of cloudlets in an MEC environment. To meet the end-to-end delay requirement of a request, we jointly explored DNN partitioning and multi-thread parallelism in cloudlets for inference acceleration. Due to the NP-hardness of the problem, we devised an approximation algorithm with a provable approximation ratio for it. Finally, we evaluated the performance of the proposed algorithm by experimental simulations. Experimental results demonstrated that the proposed algorithm is promising.

References

[1] Amazon Web Services, Inc., Amazon EC2 Instance Types, 2021. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[2] M. Abadi *et al.* Tensorflow: A system for large-scale machine learning. *OSDI*, vol. 16, pp. 265 – 283, 2016.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124 – 1137, 2004.

[4] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795 – 2808, 2016.

[5] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, vol. 100, pp. 162 – 166, 2006.

[6] A. Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[7] Z. Gong *et al.* SAVE: sparsity-aware vector engine for accelerating DNN training and inference on CPUs. *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[8] C. Hu, W. Bao, D. Wang, and F. Liu. Dynamic adaptive DNN surgery for inference acceleration on the edge. *Proc. of INFOCOM'19*, pp. 1423 – 1431, IEEE, 2019,

[9] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, vol.52, no.4, pp. 615 – 629, 2017.

[10] J. Li, W. Liang, M. Huang, and X. Jia. Reliability-aware network service provisioning in mobile edge-cloud networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1545–1558, 2020.

[11] J. Li, W. Liang, W. Xu, Z. Xu, and J. Zhao. Maximizing the quality of user experience of using services in edge computing for delay-sensitive IoT applications. *Proc. of MSWiM'20*, ACM, 2020.

[12] J. Li, W. Liang, Z. Xu, and W. Zhou. Provisioning virtual services in mobile edge computing for IoT applications with multiple sources. *Proc. of LCN'20*, IEEE, 2020.

[13] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo. Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks. To appear in *IEEE Transactions on Mobile Computing*, 2020, doi: 10.1109/TMC.2020.3006507.

[14] Y. Liu *et al.* Optimizing CNN model inference on CPUs. *Proc. of USENIX Annu. Tech. Conf.*, pp. 1025 – 1040, 2019.

[15] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco. Distributed inference acceleration with adaptive DNN partitioning and offloading. *Proc. of INFOCOM'20*, pp. 854 – 863, IEEE, 2020.

[16] W. Niu *et al.* Patdnn: achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *Proc. of ASPLOS'20* , pp. 907 – 922, 2020.

[17] A. V. Nori *et al.* Proximu$: Efficiently scaling DNN inference in multi-core CPUs through near-cache compute. *arXiv preprint arXiv:2011.11695*, 2020.

[18] A. Paszke *et al.* Automatic differentiation in pytorch. *Proc. of NIPS Workshop*, 2017.

[19] Y. Sun, S. Zhou, and J. Xu. EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks. *IEEE Journal on Selected Areas in Communications*, vol.35, no.11, pp. 2637 – 2646, 2017.

[20] Y. Xiang and H. Kim. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 392 – 405, 2019.

[21] Z. Xu *et al.* Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Transactions on Parallel and Distributed Systems*, vol.32, no.4, pp.799 – 814, 2021.

[22] F. Yu *et al.* BDD100K: A diverse driving dataset for heterogeneous multitask learning. *Proc. of CVPR'20*, pp. 2633–2642, 2020.