Throughput Maximization of Delay-Aware DNN Inference in Edge Computing by Exploring DNN Model Partitioning and Inference Parallelism

Jing Li, Weifa Liang, *Senior Member, IEEE*, Yuchen Li, Zichuan Xu, *Member, IEEE*, Xiaohua Jia, *Fellow, IEEE*, and Song Guo, *Fellow, IEEE*

Abstract—Mobile Edge Computing (MEC) has emerged as a promising paradigm catering to overwhelming explosions of mobile applications, by offloading compute-intensive tasks to MEC networks 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 user request in an MEC network, the DNN inference model usually can be partitioned into two connected parts: one part is processed in the local IoT device of the request, and another part is processed in a cloudlet (edge server) in the MEC network. Also, the DNN inference can be further accelerated by allocating multiple threads of the cloudlet to 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 partitioning and multi-thread execution parallelism. Specifically, we consider the problem under both offline and online request arrival settings: a set of DNN inference requests is given in advance, and a sequence of DNN inference requests arrives one by one without the knowledge of future arrivals, respectively. We first show that the defined problems are NP-hard. We then devise a novel constant approximation algorithm for the problem under the offline setting. We also propose an online algorithm with a provable competitive ratio for the problem under the online setting. We finally evaluate the performance of the proposed algorithms through experimental simulations. Experimental results demonstrate that the proposed algorithms are promising.

Index Terms—Mobile edge computing (MEC), DNN model inference provisioning, throughput maximization, Intelligent IoT devices, approximation and online algorithms, delay-aware DNN inference, DNN partitioning, inference parallelism, computing and bandwidth resource allocation and optimization, algorithm design and analysis

1 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 [13]. Prized by the practitioners in both academia and industry, deep learning has ignited a booming of intelligent IoT devices, which pushes edge intelligence to the horizon [32]. However, due to portable size of most mobile devices, they are resource-constrained and have limited processing capabilities to support compute-intensive deep learning applications. Thus, traditional solutions to such applications are to resort to the powerful clouds for processing [20]. Since it results in a long response delay by offloading a deep neural network (DNN) inference task from an IoT device to a remote cloud, this drives new opportunities to push DNN inferences to the edge of core networks by providing real-time DNN inference services [12]. 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 [29]. With the advance of intelligent IoT devices, it is urgent to utilize both computing and

- *Jing Li and Yuchen Li are with the School of Computing, The Australian National University, Canberra, ACT 0200, Australia. E-mail: {jing.li5, yuchen.li}@anu.edu.au.*
- *Weifa Liang and Xiaohua Jia are with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong, China. E-mail: {weifa.liang, csjia}@cityu.edu.hk.*
- *Zichuan Xu is with the School of Software, Dalian University of Technology, Dalian, Liaoning 116024, China. E-mail: z.xu@dlut.edu.cn.*
- *Song Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: song.guo@polyu.edu.hk.*

Manuscript received 11 July 2021; revised 9 September 2021; accepted 2 November 2021. Date of publication 8 November 2021; date of current version 4 April 2023.
The work of Jing Li, Weifa Liang, and Yuchen Li was supported by the Australian Research Council under its Discovery Program under Grant DP200101985. The work by Weifa Liang was also supported by a Grant from City University of Hong Kong through Project No. 9380137/CS. The work of Zichuan Xu was supported in part by the National Natural Science Foundation of China under Grant 61802048 and in part by the Xinghai Scholar Program in Dalian University of Technology, China. The work by Xiaohua Jia was supported by the Research Grants Council of Hong Kong through Project No. CityU 11214316. The work of Song Guo was supported in part by the Hong Kong RGC Research Impact Fund (RIF) through Project No. R5060-19 and in part by the Hong Kong RGC General Research Fund (GRF) through Project 152221/19E.
(Corresponding author: Weifa Liang.)
Digital Object Identifier no. 10.1109/TMC.2021.3125949

storage resources of cloudlets in an MEC network for DNN inference services in a real-time manner [31].

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 [9], 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] and PyTorch [24]), or via a customized thread pool [19]. The delay-aware DNN inference service provisioning thus poses the following challenges.

For a DNN inference request with a trained DNN model and a given inference delay requirement, how to determine which cloudlet in an 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 its inference delay requirement? How many threads in its assigned cloudlet should be allocated for processing the offloaded DNN part while meeting the inference delay requirement of the request? In this paper, we will address these challenges, and devise performance-guaranteed approximation and online algorithms for the delay-aware DNN inference throughput maximization problem in an MEC network under both offline and online request arrival settings, respectively.

The novelty of the work in this paper lies in jointly exploring DNN model partitioning and inference parallelism to accelerate the DNN inference process. The admission of a set of delay-aware DNN inference requests, rather than a single delay-aware DNN inference request, is considered for the first time in MEC. Also, to speed up the inference process in MEC, the multi-thread concept is adopted in DNN inferences, and performance-guaranteed approximation and online algorithms for the defined problem under both offline and online request arrival settings are proposed and analyzed.

The main contributions of this paper are given as follows. We first study a delay-aware DNN inference throughput maximization problem in an edge computing environment under an offline setting of request admissions, 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 inference delay requirement. We then show the NP-hardness and devise an approximation algorithm for this offline setting. We also deal with the problem under the online setting of request arrivals, where a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals, for which we develop an online algorithm with a provable competitive ratio. We finally conduct experimental simulations to evaluate the performance of the proposed algorithms. Experimental results demonstrate that the proposed algorithms are promising.

The rest of the paper is organized as follows. Section 2 reviews the related work on DNN inference service provisioning in an edge computing environment. Section 3 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 4 devises an approximation algorithm for the delay-aware DNN inference problem when all requests are given in advance. Section 5 proposes an online algorithm for the dynamic delay-aware DNN inference problem. Section 6 evaluates the proposed algorithms empirically, and Section 7 concludes the paper.

2 Related Work

Mobile Edge Computing (MEC), as an emerging paradigm, delivers a promising solution to delay-aware task offloading services, complementary to traditional cloud computing. The delay-aware task offloading in MEC environments has drawn much attention in recent years. For example, Bozorgchenani *et al.* [3] proposed an evolutionary algorithm to address a task offloading problem in an MEC network, with the aim to minimize not only the task processing delay but also the energy consumption of the processing. Li *et al.* [15], [16] studied the service provisioning of delay-sensitive IoT applications to maximize the user service satisfaction in an MEC network, by devising efficient approximation and online algorithms. Liu *et al.* [18] dealt with the stochastic arrivals of heterogeneous tasks in a three-layer MEC network, and developed an online algorithm to shorten the inference delay of each task processing.

Recently, much effort focused on investigating DNN inference accelerations through task offloading in MEC environments. Mohammed *et al.* [20] devised a novel DNN partitioning scheme in an MEC network, and applied matching theory to distribute the DNN parts to edge servers, with the aim to minimize the total computation time. Tang *et al.* [27] considered the admissions of DNN inference requests of multiple users with the assistance of an edge server to minimize the maximum end-to-end delay among users, by developing an efficient algorithm to achieve the optimal solution. Xu *et al.* [29] 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. Zeng *et al.* [31] introduced the cooperation of multiple edge devices with heterogeneous computing capacities for DNN inference, and studied the dynamic DNN workload partitioning and the workload assignment optimization over the edge devices, with the aim to minimize system energy consumption. Kang *et al.* [12] proposed a promising DNN partitioning strategy between a mobile device and a cloud to optimize the experienced inference service delay and the energy consumption, based on the DNN layer granularity. Their strategy, however, only works for chain-topology DNNs. Hu *et al.* [9]

studied the DNN partitioning problem in an integrated network consisting of edge servers and a cloud by proposing an optimal DNN partitioning strategy DSL, with the aim to minimize the delay for processing one video frame when the network workload is light. They reduced the problem to a minimum cut problem. Unfortunately, the optimal partitioning claim is suspicious, which will be detailed later.

There are approaches for multi-threading DNN inference acceleration, e.g., adopting the frameworks such as TensorFlow [2] and PyTroch [24]. For example, Liu *et al.* [19] designed a customized thread pool for convolutional neural network (CNN) inference through multi-threading on multi-core central processing units (CPUs). Nori *et al.* [22] leveraged a multi-level cache hierarchy to improve the performance of inference parallelism with multi-core CPUs. Furthermore, the resource allocation problem under DNN inference parallelism has also been investigated by researchers [23], [28]. Niu *et al.* [23] 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.* [28] 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 [23], [28] cannot be applied in this work, since important constraints in the problem such as the inference delay requirements of requests have not been considered.

Inspired by the work in [9], we approach our problem in this paper by reducing it to another problem in an auxiliary graph. The authors in [9] claimed that "an optimal DNN partitioning strategy" DSL was proposed to minimize the delay for processing a single video frame, by reducing their problem into a minimum cut problem. However, they did not provide any formal proof of their claim. In fact, their claim is incorrect, which is explained as follows. In their auxiliary graph construction, there is a pair of directed edges (s, v_i) and (v_i, t) that are from source s to a layer node v_i and from the layer node v_i to the destination t 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 node v_i on the cloud and the IoT device. Since the processing capability in the IoT device is no greater than that of the cloud, the capacity of edge (s, v_i) (the processing delay) is no greater than that of edge (v_i, t), this implies that the entirely fractional flow from source s along edge (s, v_i) can reach the destination t through edge (v_i, t). Then, all edges (s, v_i) in the auxiliary graph will be saturated, and none of the edges (v_i, t) will be saturated, by applying any minimum-cut algorithm. Thus, the minimum-cut algorithm cannot deliver an optimal DNN partition, instead, it will deliver the entire DNN model to be offloaded to the edge server. This error unfortunately cannot be fixed. In addition, the initial data (the raw input data of the DNN model) uploading time has not been included in their consideration, because there is not any uploading edge in the minimum cut delivered by their algorithm. The essential differences of our work in this paper from the study in [9] lie in the following aspects. (i) They focused on a single DNN model inference by minimizing the inference delay on a cloud or an MEC network. In contrast, we deal with a set of delay-aware DNN inference requests by admitting as many requests as possible while meeting their inference delay requirements. (ii) They assumed that the data transmission bandwidth from the IoT device of a DNN inference request is fixed and 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 vary, which are determined by both the transmission power of the IoT device and the distances between the IoT device and the cloudlets. (iii) We introduce the multi-thread concept to speed up the DNN inference process of each request in a cloudlet in order to meet the inference delay requirement of the DNN request.

On the other hand, unlike those aforementioned studies, in this paper we deal with the delay-aware DNN inference service provisioning in an edge computing environment under both offline and online settings of request arrivals. We jointly perform DNN partitioning and explore inference parallelism through deploying multiple threads to allocate offloaded tasks to different cloudlets. We aim to maximize the number of DNN inference requests admitted while meeting their inference delay requirements, subject to computing capacities on cloudlets. It must be mentioned that this paper is an extension of a conference paper [14]. The conference version of this paper only dealt with the design and analysis of the approximation algorithm for the problem of concern for admissions of a set of DNN inference requests under the offline setting. In this extended version, we consider the problem under the dynamic request arrival setting, where a sequence of DNN inference requests arrives one by one without the knowledge of future arrivals, and devise an online algorithm with provable competitive ratio for this dynamic DNN inference request admissions.

3 Preliminaries

In this section we first introduce the system model, we then provide notions and notations. We finally define the problems precisely.

3.1 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 [17]. For the sake of convenience, denote by N the set of cloudlets too. Since the wireless transmission range of each IoT device is fixed, the number of cloudlets the device can reach is limited [26]. 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 an MEC network, 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, DNN inference requires much less

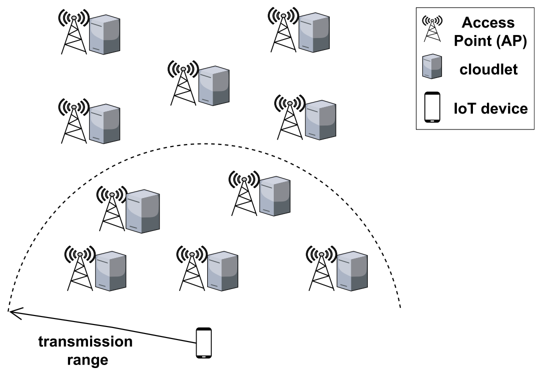


Fig. 1. An example of a geographical area with 10 APs, each of APs is co-located with a cloudlet. There is an IoT device which can offload its task to any cloudlet within its transmission range if the cloudlet has sufficient computing resource to process its offloaded task.

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 [22], due to their high availability and efficient scalability [8]. 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 [19]. We 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.

3.2 User Requests and the DNN Model

There is a set R of delay-aware DNN inference requests from different user IoT devices. Each request $r_i \in R$ has an inference delay requirement D_i [21], 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. An example of a DNN is given in Fig. 2.

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 layer nodes in 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.

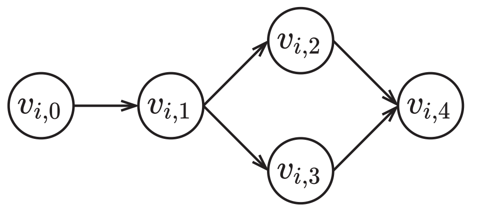

$$\begin{tikzcd} & v_{i,2} \arrow[dr] & \\ v_{i,0} \arrow[r] & v_{i,1} \arrow[u] \arrow[d] & v_{i,4} \\ & v_{i,3} \arrow[ur] & \end{tikzcd}$$

Fig. 2. An example of a DNN that consists of 4 layers.

3.3 Inference Delay Model

Because the result of a DNN inference usually is small, the downloading time of the inference result is negligible [29]. The inference delay of a DNN inference request usually consists of (1) the processing delay of the DNN model part on the local IoT device; (2) the data transmission delay from the local IoT device to the assigned cloudlet; and (3) the processing delay of the DNN model part on the cloudlet, which are defined as follows.

The Processing Delay of the DNN Model Part 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

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

Since V_i^{loc} is 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

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

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 [29]. 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 [7], i.e.,

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

where B_n is the bandwidth of the AP co-located with cloudlet n, P_i is the transmission power of the IoT device of r_i, $dist_{i,n}$ is the distance between the IoT device of r_i and the AP co-located with cloudlet n [6], σ^2 is the noise power, and β is a path loss factor with $\beta = 2$ or 4 for short or long distance, respectively [6].

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 layer set whose outputs will be uploaded and transferred to cloudlet n. Then the transmission delay of the output data by V_i^{tran} of request r_i is

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

The Processing Delay of the DNN Model Part on the Cloudlet. The inference processing delay of the offloaded layers in set V_i^{mec} in a cloudlet is assumed to be proportional to the number of allocated threads (e.g., 1 thread versus 2 threads) to this offloaded part. Furthermore, assume that there are at most K threads to be allocated to each offloaded DNN part in a cloudlet. All threads in any cloudlet n are assumed to 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 the offloaded part of 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 $v_{i,j}$ assigned to cloudlet n with $k_{i,n}$ allocated threads 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

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

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 inference delay requirement D_i, we must have

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

3.4 Problem Definition

Definition 1. *Given a set N of cloudlets co-located with APs in a geographical area, and a set of delay-aware DNN 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 inference delay requirement D_i, assuming that the model has been stored in both its local IoT device and cloudlets. 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 inference delay requirements, subject to computing capacity on each cloudlet in N.*

Definition 2. *Given a set N of cloudlets co-located with APs in a geographical area, and a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals, each DNN inference request r_i issued from an IoT device has a DNN inference model G_i with the inference delay requirement D_i, assuming that each DNN inference model is stored in both its local IoT device and cloudlets. 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 threads in the cloudlet for inference process, the* dynamic delay-aware DNN inference throughput maximization problem *in an MEC environment is to maximize as many DNN inference requests admitted as possible without the knowledge of future arrivals, while meeting their inference delay requirements, subject to computing capacity on each cloudlet in N.*

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

Proof. The NP-hardness of the problem is shown through a reduction from an NP-hard problem - the maximum profit generalized assignment problem (GAP) [5], which is defined as follows. Given a set of items and a set of bins, each bin b has a capacity $cap(b)$, and a profit $p(a, b)$ can be collected by packing item a to bin b with size $size(a, b)$. The maximum profit GAP is to maximize the total profit by packing as many items as possible to the bins, subject to the capacities of bins.

We consider a special case of the delay-aware DNN inference throughput maximization problem, where the transmission delay is neglectable, and the entire DNN model of each request will be offloaded to a cloudlet for the inference process. We also assume that each cloudlet allocates one thread for each request, i.e., $K = 1$. We then can calculate the inference delay if request r_i is offloaded to cloudlet $n \in \mathbb{N}(r_i)$. Each bin b_n corresponds to a cloudlet n with the capacity of C_n and each item r_i corresponds to a request r_i. If the inference delay requirement of request r_i can be met by offloading its DNN model to cloudlet n, the profit of packing item r_i to bin b_n is 1, and 0 otherwise. The delay-aware DNN inference problem under this special case is to maximize the number of

TABLE 1
Table of Symbols

Notations	Descriptions
R	a set of DNN inference requests from different user IoT devices
N	a set of cloudlets, each of which is co-located with an Access Point (AP).
$\mathbb{N}(r_i)$	the set of cloudlets located in the data transmission range of the IoT device of request r_i
C_n	the computing capacity of cloudlet n, measured by the number of threads (CPU cores) in it
$G_i = (V_i, E_i)$	the requested DNN of the request r_i, where V_i is the set of inference layers and E_i is the set of layer dependencies
$f(v_{i,j})$	the required number of floating-point operations of layer $v_{i,j}$ in the DNN G_i
V_i^{loc} and V_i^{mec}	the layer sets of request r_i executed at the local IoT device and a cloudlet, respectively
$V_i^{tran} \subset V_i^{loc}$	a layer set that the output of each layer $v_{i,j} \in V_i^{tran}$ will be sent to the cloudlet for processing
$\mathcal{N}^+(v_{i,j})$ and $\mathcal{N}^-(v_{i,j})$	the outgoing and incoming layer set of layer $v_{i,j}$, respectively
$\eta_{loc}(r_i)$ and $\eta_{mec}(n)$	the processing rate of the local IoT device of request r_i and the processing rate of a thread on cloudlet n
$d_{i,j}^{loc}$	the processing delay of layer $v_{i,j}$ on the local IoT device of request r_i
$d_{loc}(V_i^{loc})$	the total processing delay of the layer set V_i^{loc} on the local IoT device of request r_i
$O_{i,j}$ and P_i	the output data size of layer $v_{i,j}$ in DNN G_i and the transmission power of the IoT device of request r_i
$dist_{i,n}$	the distance between the IoT device of request r_i and AP co-located with cloudlet n
σ^2 and β	the noise power and the path loss factor
$\lambda_{i,n}$	the data transmission rate from the IoT device of user request r_i to cloudlet $n \in \mathbb{N}(r_i)$
$d_{i,j}^{tran}(n)$	the transmission delay of transmitting the output of layer $v_{i,j}$ of request r_i to cloudlet $n \in \mathbb{N}(r_i)$
$d_{tran}(V_i^{tran}, n)$	the total transmission delay of the output data by V_i^{tran} of request r_i to cloudlet $n \in \mathbb{N}(r_i)$
K	at most K threads can be allocated to accelerate the processing of each offloaded DNN part in a cloudlet
$k_{i,n}$	the number of allocated threads in cloudlet $n \in \mathbb{N}(r_i)$ to process the DNN part of request r_i
$k_{i,n}^{min}$	the minimum number of threads needed on cloudlet n to meet the delay requirement of request r_i
$d_{i,j}^{mec}(n, k_{i,n})$	the processing delay of layer $v_{i,j}$ in cloudlet n with $k_{i,n}$ allocated threads for request r_i
$d_{mec}(V_i^{mec}, n, k_{i,n})$	the processing delay of the offloaded layer set V_i^{mec} of request r_i in cloudlet n with $k_{i,n}$ allocated threads
$d_{d2d}(V_i^{loc}, V_i^{mec}, n, k_{i,n})$	the end-to-end delay of offloading layer set V_i^{mec} to cloudlet $n \in \mathbb{N}(r_i)$ with $k_{i,n}$ allocated threads for request r_i
D_i	the end-to-end inference delay requirement of request r_i
$G'_{i,n,k_{i,n}} = (V'_i, E'_i)$	the constructed auxiliary flow network for offloading the DNN part of r_i to cloudlet n with $k_{i,n}$ allocated threads
M and M^*	a potential cut and the minimum cut on the auxiliary flow network G'
$C_n(i)$	the residual computing capacity (the available number of threads) in cloudlet n when request r_i arrives
$w_n(i)$ and $\psi_n(i)$	the usage cost and the normalized usage cost of cloudlet n when request r_i arrives
α	a tuning parameter that reflects the sensitivity of the computing resource usage ratio of any cloudlet

admitted requests, subject to computing capacities on cloudlets. It can be seen that this special problem is equivalent to the maximum profit GAP. Thus, the delay-aware DNN inference problem is NP-hard, due to the NP-hardness of the maximum profit GAP. □

For the sake of convenience, the symbols used in this paper are summarized in Table 1.

4 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 $n \in N$ with $k_{i,n}$ allocated threads, $1 \leq k_{i,n} \leq K$. If such an assignment is feasible (i.e., meeting its inference 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 inference 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, approximation ratio, and time complexity of the proposed approximation algorithm.

4.1 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 inference delay by offloading part of the DNN model to cloudlet n with $k_{i,n}$ allocated threads. If this minimum inference delay meets the inference delay requirement D_i, the DNN partitioning and the DNN part offloading are feasible.

In the following, we construct the auxiliary graph $G'_{i,n,k_{i,n}}$ for the DNN model partitioning and offloading 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 $E'_i = \{(s, v_i), (v_{i,j}, v'_{i,j}) \mid v_{i,j} \in V_i\} \cup \{(v_{i,j}, t) \mid v_{i,j} \in V_i \setminus \{v_{i,0}\}\} \cup \{(v'_{i,j}, v_{i,l}), (v_{i,l}, v_{i,j}) \mid (v_{i,j}, v_{i,l}) \in E_i\}$. 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 $d_{i,j}^{mec}$ of layer $v_{i,j}$ on cloudlet n with $k_{i,n}$ allocated threads, i.e., $w(s, v_{i,j}) = d_{i,j}^{mec}$.

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 $d_{i,j}^{loc}$ of layer $v_{i,j}$ on the local IoT device of request r_i, i.e., $w(v_i, t) = d_{i,j}^{loc}$. 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., $w(v_{i,j}, v'_{i,j}) = d_{i,j}^{tran}$.

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 $d_{i,j}^{tran}$ 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 $\mathcal{N}^+(v_{i,j})$, $d_{i,j}^{tran}$ 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. 3. 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., $V_i^{loc} = V_s \cap V_i = \{v_{i,0}, v_{i,1}\}$, 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 with $k_{i,n}$ allocated threads, i.e., $V_i^{mec} = V_t \cap V_i = \{v_{i,2}, v_{i,3}, v_{i,4}\}$.

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

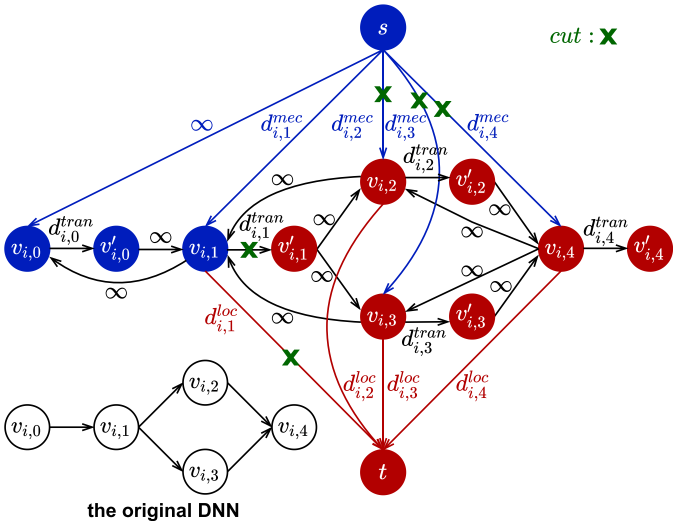


Fig. 3. An illustrative example of the auxiliary flow network $G'_{i,n,k_{i,n}}$ driven from a DNN (consisting of 4 layers) of request r_i, by assigning part of the DNN to cloudlet n with $k_{i,n}$ allocated threads for the DNN inference process.

4.2 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 $k_{i,n}^{min}$ of allocated threads in cloudlet n for request r_i with $1 \le k_{i,n}^{min} \le k_{i,n}$ needs to be identified while still meeting the delay requirement. This can be achieved by binary search on the value range $[1, K]$ of $k_{i,n}^{min}$ by constructing no more than $\lceil \log K \rceil$ auxiliary graphs.

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 inference 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 $k_{i,n}^{min}$ 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 $1 \le i \le |R|$. If item i can be assigned to bin b_n with $k_{i,n}^{min}$ allocated threads while meeting its delay requirement, it has a profit of $p(i,n)$ $(=1)$ with a size of $k_{i,n}^{min}$; otherwise, its profit is $p(i,n) = 0$ with a size of ∞, the problem then is 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 [5].

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

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 and a set of requests R.

Output: Maximize the network throughput by admitting as many DNN inference requests as possible.

```
1:  for each request $r_i \in R$ do
2:    for each cloudlet $n \notin \mathbb{N}(r_i)$ do
3:      $k^{min}_{i,n} \leftarrow \infty; p(i,n) \leftarrow 0;$
4:    end for
5:    for each cloudlet $n \in \mathbb{N}(r_i)$ do
6:      Construct auxiliary graph $G'_{i,n,K}$;
7:      Calculate the inference delay $d_{d2d}(r_i, n, K)$ by finding a
        minimum cut in $G'_{i,n,K}$;
8:      if $d_{d2d}(r_i, n, K) > D_i$ then
9:        $k^{min}_{i,n} \leftarrow \infty$; $p(i,n) \leftarrow 0$; /* the delay requirement of
          request $r_i$ cannot be met when assigned to cloudlet
          $n$ with $K$ threads.*/
10:     else
11:       $k_l \leftarrow 1$; $k_r \leftarrow K$; /* Use binary search to find a mini-
          mum number $k^{min}_{i,n}$ of threads to meet the delay
          requirement $D_i$ of request $r_i$ when it is assigned to
          cloudlet $n$/*
12:       while $k_l \le k_r$ do
13:         $k_m \leftarrow \lfloor (k_l + k_r)/2 \rfloor$;
14:         Construct auxiliary graph $G'_{i,n,k_m}$;
15:         Calculate the inference delay $d_{d2d}(r_i, n, k_m)$ by find-
            ing a minimum cut in $G'_{i,n,k_m}$;
16:         if $d_{d2d}(r_i, n, k_m) \le D_i$ then
17:           $k_r \leftarrow k_m - 1$;
18:         else
19:           $k_l \leftarrow k_m + 1$;
20:         end if
21:       end while
22:       $k^{min}_{i,n} \leftarrow k_l; p(i,n) \leftarrow 1;$
23:     end if
24:   end for
25: end for
26: 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 $k^{min}_{i,n}$ and a
    profit $p(i,n)$ if assigned to bin $b_n$;
27: An approximate solution $\mathbb{A}$ is obtained, by invoking the
    approximation algorithm for the maximum profit GAP in [5];
28: return Solution $\mathbb{A}$ to the delay-aware DNN inference
    problem.
```

4.3 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 approximation algorithm.

Lemma 1. *Given a DNN $G_i = (V_i, E_i)$, its layer set V_i is partitioned into two disjoint subsets V_i^{loc} and V_i^{mec}, where V_i^{loc} and V_i^{mec} are executed in the local IoT device of request r_i and a cloudlet in edge computing, respectively. For each layer 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 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 the cut M, and $v_{i,j}$ is always 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, as 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 of $G'_{i,n,k_{i,n}}$. 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. In the following, we will show how to derive a feasible cut M in $G'_{i,n,k_{i,n}}$ from a potential DNN partitioning $\{V_i^{loc}, V_i^{mec}\}$ in G_i.

Similar to Lemma 3, we first 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., 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. In contrast to Lemma 4, now we need to show that the derived DNN partitioning is feasible, given the minimum cut M^*.

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 Lemmas 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 $v_{i,j} \in V_i^{tran}$ with $(v_{i,j}, v'_{i,j}) \notin M^*$, and there is a layer dependency $(v_{i,j}, v_{i,l})$ with $v_{i,j} \in V_i^{tran} \subset V_i^{loc}$ 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 inference delay.*

Lemma 6 can be derived from Lemmas 4 and 5.

Theorem 2. *Given a set N of cloudlets co-located with APs in a monitoring area, and a set R of delay-aware DNN inference requests with inference delay requirements, there is a $\frac{1}{2+\epsilon}$−approximation algorithm, `Algorithm` 1, for the delay-aware DNN inference throughput maximization problem, which takes $O(|R| \cdot |N| \cdot \lceil \log K \rceil \cdot (|V|_{max} + |E|_{max}) \cdot |V|^2_{max} + |N| \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$ 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 inference delay of a request r_i by Lemma 6, and a minimum number of allocated threads for request r_i to meet its inference 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 [4]. Also, it takes $O(|N| \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$ time for the maximum GAP, by the approximation algorithm in [5]. Thus, the time complexity of `Algorithm` 1 is $O(|R| \cdot |N| \cdot \lceil \log K \rceil \cdot (|V|_{max} + |E|_{max}) \cdot |V|^2_{max} + |N| \cdot |R| \cdot \log \frac{1}{\epsilon} + \frac{|N|}{\epsilon^4})$. □

5 Online Algorithm for the Dynamic Delay-Aware DNN Inference Throughput Maximization Problem

In this section, we deal with dynamic admissions of delay-aware DNN inference requests, by assuming that a sequence of requests arrives one by one without the knowledge of future request arrivals. We propose an online algorithm with a provable competitive ratio for the dynamic delay-aware DNN inference throughput maximization problem.

5.1 Online Algorithm

In the proposed online algorithm, we adopt the similar strategy as we did for `Algorithm` 1. That is, for each incoming request r_i, we first identify the minimum number $k_{i,n}^{min}$ of allocated threads on each cloudlet $n \in \mathbb{N}(r_i)$ to meet its delay requirement, where the value of $k_{i,n}^{min}$ is obtained by binary search on the value range $[1, K]$, and we construct no more than $\lceil \log K \rceil$ auxiliary graphs to find $k_{i,n}^{min}$.

Let $C_n(i)$ be the residual computing capacity (the available number of threads) in cloudlet $n \in \mathbb{N}(r_i)$ when request r_i arrives, and $C_n(1) = C_n$ initially. If request r_i is admitted, $k_{i,n}^{min}$ threads in cloudlet n are allocated to process its offloaded DNN layers to meet its inference delay requirement D_i, then $C_n(i) = C_n(i) - k_{i,n}^{min}$. We model the *usage cost* $w_n(i)$ of cloudlet n by an exponential function when considering request r_i admission as follows.

$$w_n(i) = C_n\left(\alpha^{1-\frac{C_n(i)}{C_n}} - 1\right), \tag{10}$$

where $(1 - \frac{C_n(i)}{C_n})$ is the computing resource usage ratio of cloudlet n, and $\alpha > 1$ is a tuning parameter that reflects the sensitivity of the computing resource usage ratio of any cloudlet.

Algorithm 2. An Online Algorithm for the Dynamic Delay-Aware DNN Inference Throughput Maximization Problem

```
Input: An MEC network consisting of a set $N$ of cloudlets, and a sequence of requests arriving one by one without the knowledge of future arrivals.
Output: Maximize the number of admitted requests.
1: $\mathbb{A} \leftarrow \emptyset$; /* the solution */
2: while a request $r_i$ arrives do
3:   $Q_i \leftarrow \mathbb{N}(r_i)$; /* the candidate cloudlet set for $r_i$ */
4:   for each cloudlet $n \in Q_i$ do
5:     Construct auxiliary graph $G'_{i,n,K}$;
6:     Calculate the inference delay $d_{d2d}(r_i, n, K)$ by finding a minimum cut in $G'_{i,n,K}$;
7:     if $d_{d2d}(r_i, n, K) > D_i$ then
8:       $k_{i,n}^{min} \leftarrow \infty$;
9:     else
10:      $k_l \leftarrow 1$; $k_r \leftarrow K$;
11:      while $k_l \leq k_r$ do
12:        $k_m \leftarrow \lfloor (k_l + k_r)/2 \rfloor$;
13:        Construct auxiliary graph $G'_{i,n,k_m}$;
14:        Calculate the inference delay $d_{d2d}(r_i, n, k_m)$ by finding a minimum cut in $G'_{i,n,k_m}$;
15:        if $d_{d2d}(r_i, n, k_m) \leq D_i$ then
16:          $k_r \leftarrow k_m - 1$;
17:        else
18:          $k_l \leftarrow k_m + 1$;
19:        end if
20:      end while
21:      $k_{i,n}^{min} \leftarrow k_l$;
22:    end if
23:    if $k_{i,n}^{min} > C_n(i)$ then
24:      $Q_i \leftarrow Q_i \setminus \{n\}$; /* cloudlet $n$ has no sufficient number of residual threads for request $r_i$ */
25:    end if
26:  end for
27:  if $Q_i = \emptyset$ then
28:    Reject request $r_i$;
29:  else
30:    Identify the cloudlet $n' \in Q_i$ with the minimum normalized usage cost $\psi_{n'}(i)$, by Eq. (11);
31:    if $\psi_{n'}(i) > |N|$ then
32:      Reject request $r_i$;
33:    else
34:      Admit request $r_i$ by assigning the DNN part of $r_i$ to cloudlet $n'$, and $\mathbb{A} \leftarrow \mathbb{A} \cup \{r_i\}$;
35:    end if ;
36:  end if ;
37: end while
38: return Solution $\mathbb{A}$ to the dynamic delay-aware DNN inference throughput maximization problem.
```

The *normalized usage cost* $\psi_n(i)$ of cloudlet n then is defined as

$$\psi_n(i) = \frac{w_n(i)}{C_n} = \alpha^{1-\frac{C_n(i)}{C_n}} - 1. \tag{11}$$

When request r_i arrives, we partition its DNN model G_i and calculate the resulting inference delay if $k_{i,n}$ threads in cloudlet $n \in \mathbb{N}(r_i)$ are allocated for request r_i, with $1 \leq k_{i,n} \leq K$, by finding the minimum cut in the constructed auxiliary graph. We then identify a minimum number $k_{i,n}^{min}$ of threads on each cloudlet $n \in \mathbb{N}(r_i)$ to meet the inference delay requirement of request r_i, by binary search on the value range $[1, K]$ of $k_{i,n}^{min}$, which can be achieved by constructing no more than $\lceil \log K \rceil$ auxiliary graphs. Let $Q_i \subset \mathbb{N}(r_i)$ be the set of candidate cloudlets for request r_i, where each cloudlet $n \in Q_i$ has sufficient computing resource (the number of threads $k_{i,n}^{min}$) to meet the inference delay requirement of request r_i, i.e., $C_n(i) \geq k_{i,n}^{min}$.

Request r_i will be rejected if Q_i is empty, i.e., the delay requirement of request r_i cannot be met by assigning r_i to any cloudlet in the MEC network within its transmission range. Otherwise, a candidate cloudlet $n' \in Q_i$ with the minimum normalized usage cost is identified by Eq. (11). However, request r_i can still be rejected if its demanded computing resource is too costly. Therefore, an admission control policy to determine the admission of request r_i is proposed as follows: request r_i will be rejected if the normalized usage cost of cloudlet n' is greater than $|N|$, i.e., $\psi_{n'}(i) > |N|$, where N is the set of cloudlets.

The detailed online algorithm for the dynamic delay-aware DNN inference throughput maximization problem is given in `Algorithm` 2.

5.2 Algorithm Analysis

In the following, we analyze the competitive ratio and time complexity of the proposed online algorithm, `Algorithm` 2.

Lemma 7. *Given an MEC network consisting of a set N of cloudlets, a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals. Assume that at most K threads on a cloudlet can be allocated to accelerate the DNN inference of each request. Let $\mathcal{A}(i)$ be the set of requests admitted by* `Algorithm` *2 prior to the arrival of request r_i. When request r_i arrives, the sum of the usage costs of all cloudlets in N is*

$$\sum_{n \in N} w_n(i) \leq 2 \cdot K \cdot |N| \cdot |\mathcal{A}(i)| \cdot \log_2 \alpha. \tag{12}$$

The proof of Lemma 7 is given in Section 1 of the supplementary material file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2021.3125949.

Lemma 8. *Given an MEC network consisting of a set N of cloudlets, a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals. Assume that at most K threads can be allocated on any cloudlet to accelerate the DNN inference of a request. Let $\mathcal{B}(i)$ be the set of requests admitted by the optimal solution but rejected by* `Algorithm` *2 prior to the arrival of request r_i. Each request $r_{i'} \in \mathcal{B}(i)$ is assumed to be assigned to cloudlet $n^*_{i'}$ in the optimal solution. Then, for each request $r_{i'} \in \mathcal{B}(i)$, we have*

$$\psi_{n^*_{i'}}(i') > |N|, \tag{13}$$

when $2|N|+2 \leq \alpha \leq 2^{\frac{C_{min}}{K}}$, and $C_{min} = \min_{n\in N}\{C_n\}$ is the minimum computing capacity of a cloudlet.

The proof of Lemma 8 is given in Section 2 of the supplementary material file, available online.

Theorem 3. *Given a set N of cloudlets co-located with $|N|$ APs, each cloudlet $n \in N$ has computing capacity c_v in terms of the number of threads, a sequence of delay-aware DNN inference requests arrives one by one without the knowledge of future arrivals. Assuming at most K threads in a cloudlet can be allocated to accelerate the DNN inference of each request, there is an online algorithm for the dynamic delay-aware DNN inference throughput maximization problem, `Algorithm` 2, with a competitive ratio of $O(\log|N|)$. The algorithm takes $O(|N| \cdot \lceil \log K \rceil \cdot (|V_{max}| + |E_{max}|) \cdot |V_{max}|^2)$ time for the admission of each request when $\alpha = 2|N|+2$, where N is the set of cloudlets, $|V_{max}|$ and $|E_{max}|$ are the maximum numbers of layers and edges in a DNN among the DNN models of all requests, respectively.*

The proof of Theorem 3 is given in Section 3 of the supplementary material file, available online.

6 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms through experimental simulations. We also study the impact of important parameters on the performance of the proposed algorithms.

6.1 Experimental Settings

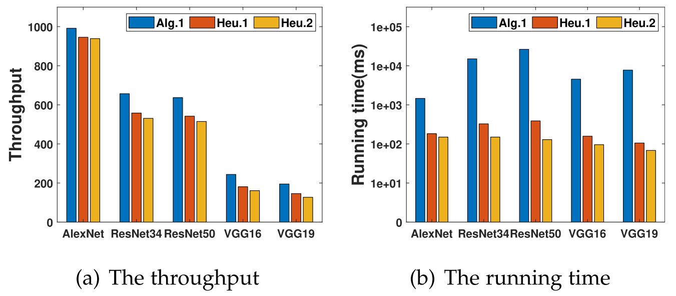
The geographical area is a 1 $km \times 1$ km square area [20] and 100 cloudlets are deployed on a regular 10×10 grid MEC network, each of which is co-located with an AP. The bandwidth of each AP varies from 2 MHz to 10 MHz [9], and each AP can provide services to a user within 100 m [26]. There are 1,000 delay-aware DNN inference requests, each of which is issued from an IoT device, and the IoT devices are randomly scattered over the defined geographical area. For DNN inferences, we here consider the well-known DNNs including `AlexNet` [10], `ResNet34`, `ResNet50` [11], `VGG16`, and `VGG19` [25]. Each request contains an image for inference, which is extracted from the videos in the self-driving dataset BDD100K [30]. The transmission power of each IoT device is randomly drawn between 0.1 Watt and 0.5 Watt [6], [26], noise power σ^2 is set as 1×10^{-10} Watt, and the path loss factor β is set as 4 [6]. The inference 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 assume that each CPU core corresponds to a thread, as applying hyper-threading technique (generate two threads per CPU core) may affect the performance due to additional context switching [19]. 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 that varies from 2.5 GHz to 3 GHz in different cloudlets [1], while the clock speed of each IoT device ranges from 0.5 GHz to 1 GHz [6]. Each cloudlet or IoT device is assumed to conduct 4 floating-point operations per cycle [20]. The parameter ϵ in `Algorithm` 1 is set as 0.5 by Theorem 2. For `Algorithm` 2, the parameter α in Eq. (11) is set as $2|N|+2$ by Theorem 3. The value in each figure is the mean of the results out of 15 different runs of MEC network instances of the same size, where the throughput refers to the number of delay-aware DNN inference requests admitted by the system among all requests in a given set or a sequence of requests for a finite monitoring period. 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 otherwise specified, the above parameters are adopted by default.

To evaluate the performance of `Algorithm` 1 (referred to as `Algorithm 1`) for the delay-aware DNN inference throughput maximization problem, we here introduce two heuristic algorithms as benchmarks: algorithms `Heu.1_off` and `Heu.2_off`. Algorithm `Heu.1_off` is based on an existing DNN partitioning strategy *Neurosurgeon* [12], which, however, only works for chain-topology DNNs. Therefore, we preprocess each DAG-topology DNN by a topological sorting approach as did in [9], and then adopt Neurosurgeon to partition the DNN between the local IoT device and a cloudlet. For each request r_i, we can find the minimum number $k_{i,n}^{min}$ of needed threads among all cloudlets to meet its delay requirement. Algorithm `Heu.1_off` then admits a request r_i with the minimum number $k_{i,n}^{min}$ of threads among all requests, iteratively, until no more request can be admitted. Algorithm `Heu.2_off` offloads the entire DNN model of a request to its nearest cloudlet with the number of needed threads to meet its inference delay requirement. Similarly, algorithm `Heu.2_off` admits a request with the minimum number of needed threads among all requests, iteratively, until no more requests can be admitted.

To investigate the performance of `Algorithm 2` (referred to as `Algorithm` 2) for the dynamic delay-aware DNN inference throughput maximization problem, two heuristic algorithms `Heu.1_on` and `Heu.2_on` are proposed, which are the online versions of algorithms `Heu.1_off` and `Heu.2_off`, respectively, i.e., algorithm `Heu.1_on` greedily admits each incoming request by offloading the request to the cloudlet with the minimum number of threads needed among all cloudlets, by the modified strategy - Neurosurgeon. In contrast, algorithm `Heu.2_on` greedily admits each incoming request by offloading its entire DNN model to the nearest cloudlet of the request that has sufficient numbers of threads. Either algorithm `Heu.1_on` or algorithm `Heu.2_on` rejects an incoming request if no cloudlet has the number of threads needed.

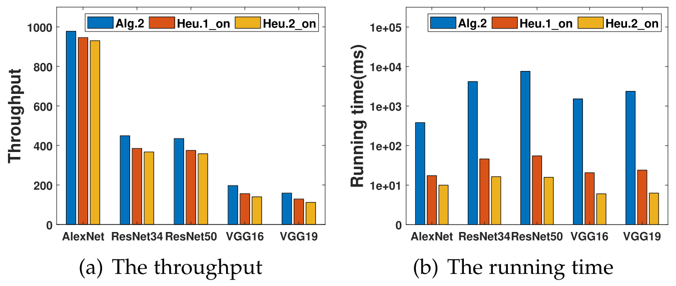
6.2 Performance Evaluation of the Proposed Algorithm for the Delay-Aware DNN Inference Throughput Maximization Problem

We first evaluated the performance of algorithm `Algorithm 1` against algorithms `Heu.1_off` and `Heu.2_off` with 1,000 requests, over the considered DNNs, respectively. Fig. 4 shows the throughput and running time delivered by different algorithms. It can be seen from Fig. 4a that algorithm `Algorithm 1` admits more requests than algorithms `Heu.1_off` and `Heu.2_off` in all cases. For example, for ResNet34, algorithm `Algorithm 1` outperforms algorithms `Heu.1_off` and `Heu.2_off` by 17.7% and



(a) The throughput (b) The running time

Fig. 4. Performance of different algorithms for the delay-aware DNN inference throughput maximization problem over different DNNs.



(a) The throughput (b) The running time

Fig. 6. Performance of different algorithms for the dynamic delay-aware DNN inference throughput maximization problem over different DNNs.

23.7%, respectively. This is because algorithm `Algorithm 1` establishes an efficient DNN partitioning strategy for each inference request, and makes resource-efficient decisions of request admissions, compared with the benchmark algorithms. Also, the performance of algorithm `Algorithm 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).

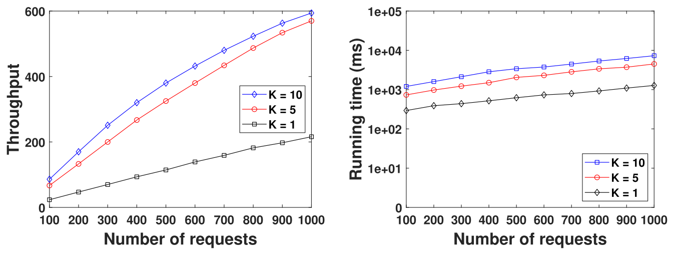
We then studied the impact of parameter K on the performance of algorithm `Algorithm 1`, by varying the number of requests from 100 to 1,000, where the requested DNN of each request is randomly drawn from the predefined DNN set. Fig. 5 illustrates the throughput and running time of algorithm `Algorithm 1` when $K = 1$, 5, and 10, respectively. From Fig. 5a, we can see that when the number of requests is 1,000, the performance of algorithm `Algorithm 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 inference delay requirement.

6.3 Performance Evaluation of the Proposed Algorithms for the Dynamic Delay-Aware DNN Inference Throughput Maximization Problem

We now investigated the performance of algorithm `Algorithm 2` against algorithms `Heu.1_on` and `Heu.2_on` for the dynamic delay-aware DNN inference throughput maximization problem, over the considered DNNs, respectively. Fig. 6 plots the throughput and running time of different algorithms. From Fig. 6a, we can see that algorithm `Algorithm 2` outperforms algorithms `Heu.1_on` and `Heu.2_on` respectively. For example, for ResNet34, algorithm `Algorithm 1` outperforms algorithms `Heu.1_off` and `Heu.2_off` by 18.7% and 21.1%, respectively. The reason is that algorithm `Algorithm 2` assigns an incoming request to a cloudlet with the minimum normalized usage cost by Eq. (11), and minimizes the number of threads needed to meet its delay requirement by an efficient DNN partitioning strategy. A well-designed admission control policy is also adopted by algorithm `Algorithm 2` to avoid resource overspending. In Fig. 6a, algorithm `Algorithm 2` over AlexNet has the best performance, among the predefined DNN set, because the inference over AlexNet requires the smallest number of floating-point operations, which is consistent with the performance behaviors in Fig. 4a.

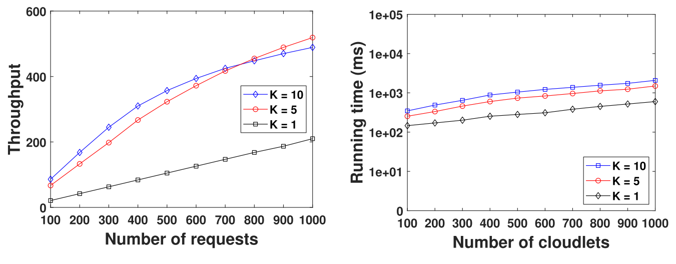
We then studied the impact of parameter K on the performance of algorithm `Algorithm 2`, by varying the number of requests from 100 to 1,000, where the requested DNN of each request is randomly drawn from the predefined DNN set. Fig. 7 depicts the performance curves of algorithm `Algorithm 2` when $K = 1$, 5, and 10, respectively. It can be seen from Fig. 7a that when the number of requests is 100, the throughput achieved by algorithm `Algorithm 2` with $K = 1$ is 23.2% of itself with $K = 10$. This is because more requests with stringent delay requirements can be admitted with a large value of K in the case of abundant network resource (with a small number of incoming requests). However, with over 800 requests, algorithm `Algorithm 2` with $K = 5$ delivers the best performance, compared with itself with $K = 1$ or 10. The rationale behind this is that when the resource is limited (with a large number of incoming requests), setting K with a reasonable value helps to maximize the throughput by reserving the resource for future request admissions.

We finally evaluated the impact of the admission control policy and parameter α on the performance of algorithm `Algorithm 2`, where $\alpha > 1$ is a tuning parameter that reflects the sensitivity of the computing resource usage ratio



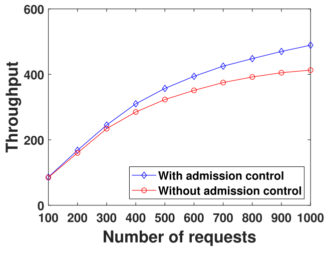
(a) The throughput (b) The running time

Fig. 5. Impact of parameter K on the performance of algorithm `Algorithm 1`.

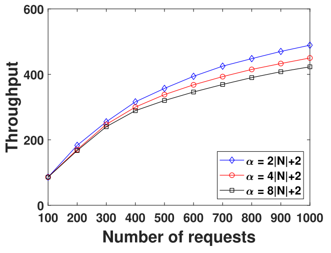


(a) The throughput (b) The running time

Fig. 7. Impact of parameter K on the performance of algorithm `Algorithm 2`.



(a) The throughput



(b) The throughput

Fig. 8. The impacts of the admission control policy and parameter α on the performance of algorithm `Algorithm 2`.

of any cloudlet. Fig. 8 shows the performance behaviors of algorithm `Algorithm 2` with and without the admission control policy. Algorithm `Algorithm 2` with the admission control policy admitted 18.4% more requests than itself without the admission control policy in the case of 1,000 requests. This can be justified by that an efficient admission control policy intends to admit requests with a small number of threads needed, therefore, the achieved throughput is maximized. Fig. 8b plots the throughput curves of algorithm `Algorithm 2` with parameter $\alpha = 2|N| + 2$, $4|N| + 2$, and $8|N| + 2$, respectively, where $|N|$ is the number of cloudlets. It can be seen from Fig. 8b that when the number of requests reaches 1,000, the performance of algorithm `Algorithm 2` with $\alpha = 8|N| + 2$ admitted 86.5% of itself with $\alpha = 2|N| + 2$. This is because the normalized usage cost increases with the increase on the value of parameter α by Eq. (11), and algorithm `Algorithm 2` intends to reject the incoming requests by the admission control policy.

7 Conclusion

In this paper, we investigated the DNN inference service provisioning with inference delay requirements in an MEC environment. We studied a delay-aware DNN inference throughput maximization problem with the aim to maximize the number of requests admitted, subject to computing capacities on cloudlets in the MEC network. To meet the inference delay requirement of each request, we jointly explored the DNN model partitioning and inference parallelism in the cloudlet for inference acceleration. We then showed the NP-hardness of the problem and devised an approximation algorithm with a provable approximation ratio for the problem. In addition, we also considered the dynamic delay-aware DNN inference throughput maximization problem where a sequence of dealy-aware DNN inference requests arrives one by one without the knowledge of future arrivals, for which we developed an online algorithm with a provable competitive ratio. We finally evaluated the performance of the proposed algorithms by experimental simulations. Experimental results demonstrated that the proposed algorithms are promising.

Acknowledgments

The authors would like to thank the three anonymous referees and the Associate Editor for their constructive comments and invaluable suggestions, which help them improve the quality and presentation of the paper greatly.

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] A. Bozorgchenani, F. Mashhadi, D. Tarchi, and S. Salinas, "Multi-objective computation sharing in energy and delay constrained mobile edge computing environments," *IEEE Trans. Mob. Comput.*, vol. 20, no. 10, pp. 2992–3005, Oct. 2021.
[4] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, pp. 1124–1137, Sep. 2004.
[5] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, pp. 162–166, 2006.
[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[7] A. Goldsmith, *Wireless Communications*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[8] Z. Gong, H. Ji, C. W. Fletcher, C. J. Hughes, S. Baghsorkhi, and J. Torrellas, "SAVE: sparsity-aware vector engine for accelerating DNN training and inference on CPUs," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 796–810.
[9] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE INFOCOM*, 2019, pp. 1423–1431.
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778.
[12] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, 2017.
[13] J. Li, W. Liang, M. Huang, and X. Jia, "Reliability-aware network service provisioning in mobile edge-cloud networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1545–1558, Jul. 2020.
[14] J. Li, W. Liang, Y. Li, Z. Xu, and X. Jia, "Delay-aware DNN inference throughput maximization in edge computing via jointly exploring partitioning and parallelism," in *Proc. LCN IEEE*, 2021, pp. 193–200.
[15] J. Li *et al.*, "Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1199–1212, May 2022.
[16] 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," in *Proc. MSWiM ACM*, 2020, pp. 113–121.
[17] J. Li, W. Liang, Z. Xu, and W. Zhou, "Provisioning virtual services in mobile edge computing for IoT applications with multiple sources," in *Proc. LCN IEEE*, 2020, pp. 42–53.
[18] T. Liu, L. Fang, Y. Zhu, W. Tong, and Y. Yang, "A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing," *IEEE Trans. Mobile Comput.*, early access, 17 Dec., 2020, doi: 10.1109/TMC.2020.3045471.
[19] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing CNN model inference on CPUs," in *Proc. USENIX Annu. Technol. Conf.*, 2019, pp. 1025–1040.
[20] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. INFOCOM*, 2020, pp. 854–863.
[21] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks," *IEEE Trans. Mobile Comput.*, early access, Jul. 2020, doi: 10.1109/TMC.2020.3006507.
[22] A. V. Nori *et al.*, "Proximu$: Efficiently scaling DNN inference in multi-core CPUs through near-cache compute," 2020, *arXiv: 2011.11695*.
[23] W. Niu *et al.*, "PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *Proc. ASPLOS*, 2020, pp. 907–922.
[24] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. NIPS Workshop*, 2017, pp. 1–4.
[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[26] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[27] X. Tang, X. Chen, L. Zeng, S. Yu, and L. Chen, "Joint multiuser DNN partitioning and computational resource allocation for collaborative edge intelligence," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9511–9522, Jun. 2021.

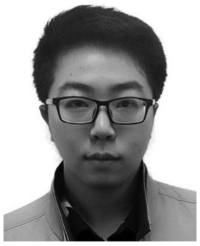
[28] Y. Xiang and H. Kim, "Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference," in *Proc. RTSS*, 2019, pp. 392–405.

[29] 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.

[30] F. Yu *et al.*, "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. CVPR*, 2020, pp. 2633–2642.

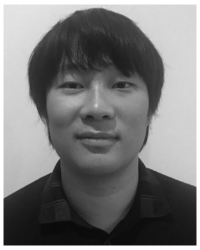
[31] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.

[32] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.



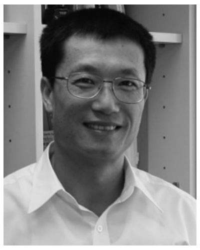
Jing Li received the BSc degree with the first class Honours in computer science from the Australian National University in 2018. He is currently working toward the PhD degree with the Research School of Computer Science, Australian National University. His research interests include mobile edge computing, network function virtualization, and combinatorial optimization.

Weifa Liang (Senior Member, IEEE) received the PhD degree in computer science from the Australian National University in 1998, the ME degree in computer science from the University of Science and Technology of China in 1989, and the BSc degree in computer science from Wuhan University, China in 1984. He is currently a professor with the Department of Computer Science, City University of Hong Kong. Prior to that, he was a professor with the Australian National University. His research interests include design and analysis of energy efficient routing protocols for wireless ad hoc and sensor networks, Mobile Edge Computing (MEC), Network Function Virtualization (NFV), Internet of Things, Software-Defined Networking (SDN), design and analysis of parallel and distributed algorithms, approximation algorithms, combinatorial optimization, and graph theory. He is currently an associate editor for the Editorial Board of *IEEE Transactions On Communications*.

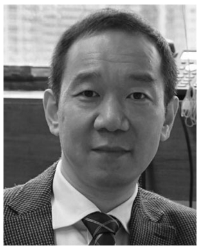


Yuchen Li received the BSc degree with the first class Honours in computer science from the Australian National University in 2018. He is currently working toward the PhD degree with the Research School of Computer Science, Australian National University. His research interests include the Internet of Things, mobile edge computing, and algorithm design.

Zichuan Xu (Member, IEEE) received the BSc and ME degrees in computer science from the Dalian University of Technology, China, in 2008 and 2011, respectively, and the PhD degree from The Australian National University, Australia, in 2016. From 2016 to 2017, he was a research associate with the Department of Electronic and Electrical Engineering, University College London, U.K. He is currently an associate professor and a PhD advisor with the School of Software, Dalian University of Technology, China. He is also a Xinghai scholar with the Dalian University of Technology, China. His research interests include cloud computing, mobile edge computing, deep learning, network function virtualization, software-defined networking, routing protocol design for wireless networks, algorithmic game theory, and optimization problems.



Xiaohua Jia (Fellow, IEEE) received the BSc and ME degrees in 1984 and 1987, respectively, from the University of Science and Technology of China, and the DSc degree in information science from the University of Tokyo in 1991. He is currently a chair professor with Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks, and mobile wireless networks. He was and is an editor of *IEEE Transactions on Parallel and Distributed Systems* from 2006 to 2009, *Journal of World Wide Web*, *Wireless Networks*, and *Journal of Combinatorial Optimization*. He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE MASS 2009, area-chair of IEEE INFOCOM 2010, TPC co-chair of IEEE GlobeCom 2010, Ad Hoc and Sensor Networking Symposium, and Panel co-chair of IEEE INFOCOM 2011.



Song Guo (Fellow, IEEE) is currently a full professor in the Department of Computing at The Hong Kong Polytechnic University. He also holds a Changjiang Chair Professorship awarded by the Ministry of Education of China. His research interests include big data, edge AI, mobile computing, and distributed systems. With many impactful papers published in top venues in these areas, he has been recognized as a Highly Cited Researcher (Web of Science) and received over 12 best paper awards from IEEE/ACM conferences, journals and technical committees. Prof. Guo is the Editor-in-Chief of IEEE Open Journal of the Computer Society. He has served on IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board of a number of prestigious international journals like IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Cloud Computing, IEEE Internet of Things Journal, etc. He has also served as chair of organizing and technical committees of many international conferences. Prof. Guo is an IEEE Fellow and an ACM Distinguished Member.