Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 8804530, 12 pages
https://doi.org/10.1155/2022/8804530

Research Article

An Adaptive Task Migration Scheduling Approach for Edge-Cloud Collaborative Inference

Boyin Zhang,[1] Yinggang Li,[1] Shigeng Zhang,[1,2] Yue Zhang,[3] and Bing Zhu[1]

[1]*School of Computer Science and Engineering, Central South University, China*
[2]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*
[3]*Hunan Provincial Key Laboratory of Network Investigational Technology, Hunan Police Academy, China*

Correspondence should be addressed to Shigeng Zhang; sgzhang@csu.edu.cn

Received 29 September 2021; Accepted 25 November 2021; Published 17 January 2022

Academic Editor: Pengfei Wang

Deep Neural Network (DNN) models have achieved excellent performance in many inference tasks and have been widely used in many intelligent applications. However, DNN models often require a lot of computational resources to complete the inference tasks, which hinders the deployment of such models to resource-constrained edge devices. In order to extend the application scenarios of DNN models, the edge-cloud collaborative inference methods, represented by model partition, have attracted much research attention in recent years. In scenarios that have multiple edge devices deployed, the edge-cloud collaborative inference method requires partial migration of tasks, but traditional scheduling methods only migrate tasks at the task level. In this paper, we propose two task scheduling methods, which can solve the problem of partial migration of tasks in multiedge scenarios. The first scheduling method is based on the optimal cutting of a single DNN. The cutting positions of all the models are the same, regardless of the influence of external factors. This method is suitable for chain and directed acyclic graph- (DAG-) type DNNs. The second scheduling method takes external factors such as congestion and queuing delay at the cloud side into consideration, which dynamically selects the cutting position of each DNN to optimize the overall delay and thus is applicable to chain DNN models. The experimental results show that, compared with the baseline method, our proposed scheduling method can reduce the delay by up to 6.48x.

1. Introduction

With the rapid development of Deep Neural Network (DNN), edge intelligence has been widely applied in the Internet of Things (IoT) scenarios in recent years [1, 2]. The growing proliferation of IoT devices with high-quality sensors will result in massive data streaming to the edge or the cloud. Edge devices deployed generally have some constraints including energy and computational capacity [3]. Thus, process data and inference only at edge ends will cause high delays that cannot meet the requirements of most applications. Cloud servers have high capacities of computation; however, transferring all data to the cloud will suffer double network delay that is intolerant in bad network conditions. Moreover, transferring whole data to the cloud takes the risk of privacy leakage [4]. Edge-cloud collaborative inference, which assigns tasks between the edge and the cloud on the basis of constraints in applications, has become a research focus nowadays.

In the edge computing environment, there are usually multiple edge devices collecting data at the same time, but the number of corresponding cloud servers is very small [5]. If a lot of tasks are generated at the edge at the same time, how to better schedule these tasks to reduce the overall delay is a problem we need to study.

The edge-cloud collaborative inference can adaptively cut the DNN according to the network bandwidth without changing the original model parameters of the DNN [6], so as to minimize the delay or maximize the throughput. For the deployment of a single edge, determine the cutting position by network bandwidth is obviously the optimal solution. However, in the real edge environment, there are

often multiple edge devices generating tasks. Due to a large number of edge tasks, the tasks that the cloud can process at the same time are limited. Therefore, how to schedule these tasks is very important. In this paper, we design a system that can perform task scheduling in multiple edge scenarios.

For a single-edge system, we determine how the DNN is cut based on the current network status and the fixed configuration parameters of the system (such as the execution time of the DNN on the edge and the cloud), through the goal of minimizing delay or maximizing throughput, find the optimal cutting point. However, in a multiedge scenario, the cutting point for minimizing the delay for a single task is not necessarily optimal for the entire system. Due to the limited processing capacity of the cloud, when the number of tasks generated at the edge is much greater than the processing capacity of the cloud, many tasks will face the problem of blocking. When the edge end completes its own inference task, it transmits the intermediate processing result to the cloud, and the cloud cannot process it in time, which causes the task to be blocked. Although the execution time of each task at the edge is optimal, the waiting time in the cloud increases the processing time of the entire system.

In this scenario, minimizing the delay of each DNN may not be the optimal solution for the entire system. If cloud task congestion is detected, the position of the DNN is appropriately adjusted so that the task it is waiting for is executed at the edge. Although the delay cannot be minimized for a single DNN, it will make the overall delay of the system less. To our knowledge, most of the scheduling schemes for the DNN model at present are scheduling the entire DNN model. Due to the dependency and complexity of the edge-cloud collaborative system scheduling problem, few researches focus on its scheduling strategy. In [7], the author studied the scheduling system in the scenario of multiple IoT devices; it suggested that each type of IoT device has different computing capabilities and designed an online scheduling (Online) system. Online can decide whether DNN is deployed locally or in the cloud, and it can also adjust the scheduling sequence. The author compares Online with first-come, first-served (FIFO) and low-bandwidth first deployment (LBF) strategies. Compared with the other two methods, Online can improve the overall quality of service. In [8], the author proposed a migration scheduling problem for DNN tasks in edge environment, gave the formal definition and evaluation criteria of the problem, and proposed a greedy algorithm and a genetic algorithm for the migration. The problem-solving is approximately optimal, which can effectively solve the migration and deployment problems of DNN.

In this paper, we designed two task scheduling strategies for edge-cloud collaborative inference. The first strategy finds the optimal cutting position of DNN models for every single task which is capable for DNN models with DAG and chain topologies. The second strategy decides cutting position on the basis of network condition to reach global optimal for all tasks, which is capable for DNN models with chain topology.

The remainder of this paper is organized as follows. In Section 2, we conduct a brief literature review of related studies. In Section 3, we define the problem to be addressed. In Section 4, we introduce our approach in detail. Section 5 shows the results of the experiments. We conclude in Section 6.

2. Related Work

Much work has been done to accelerate the inference of DNN models in IoT scenarios. In order to reduce the delay of inference at edge devices, the main research interests are divided into three aspects: model compression [9], distributed model deployment [10], and computation offloading [11].

Compression of DNN models uses technics such as prune [12], quantization [13], and knowledge distillation [14] to reduce the computation operations in model inference without significant accuracy reduction [15]. Although model compression speed up the inference on edge devices significantly, the compression itself needs lots of computation to complete and the inference of a compressed model may need specific hardware to complete. For the application environments that contain heterogeneous architecture edge devices, this method is not competent. There are multiple studies about process DNN models on resource-constrained devices effectively [16, 17]. To inference DNN models effectively on resource-constrained devices, researchers design the hardware architecture deliberately [18] and hardware-accelerating methods usually combined with model compression methods which can improve the efficiency of specific model inference significantly [19]. However, these methods are not universal to various applications compared to other methods. Edge devices in real application scenarios are generally heterogeneous in architectures; hardware accelerating requires specific computation units or ICs to execute the model effectively. Distributed deployment of DNN models can make full use of the computing resources of devices [10] and is capable for large-scale applications. However, for edge intelligence environments, the number of devices may vary dynamically; distributed deployment cannot handle this well. Moreover, distributed deployment on multiple edge devices may cause network congestion, especially under bad network conditions.

Edge-cloud collaborative inference has unique advantages over the first two technologies; it does not change the original model compared with lightweight model and distributed network deployment. Edge-cloud collaborative inference has high scalability and can be combined with the other two technologies. For edge-cloud collaborative inference, a lot of previous work has been done to achieve the purpose of reducing overall inference delay [8, 20–23] or to satisfy resource constraints [24–27] (bandwidth, power consumption, etc.) or to protect privacy [28]. Applying traditional artificial intelligence technology to edge computing, which is usually resource-constrained, researchers' ideas are mainly divided into the following three kinds: DNN model selection depending on sample [29], design lightweight DNN architectures [30] or DNN model compression [15],

and edge-cloud collaborative inference by cutting DNN models and scheduling tasks between edge and cloud [25].

In [29], the authors apply an autoencoder to compress the data transmitted to cloud platforms. Kang et al. [6] first proposed Neurosurgeon, a method that partitions the DNN model to execute on end devices and cloud platforms simultaneously to improve the efficiency of the model inference. It cuts a DNN into two parts and executes on a mobile device and the cloud platform, respectively, to accelerate the inference of the model. However, Neurosurgeon can only partition chain-like DNNs; it cannot handle DAG structure DNNs, which limits its application. What is more, Neurosurgeon does not have enough accuracy on execution time prediction because of the linear regression method it used. This leads to a nonoptimal partition of the model. Teerapittayanon et al. proposed DDNN [31] to speed up the latency of inference of a DNN model over distributed computing hierarchies, consisting of the cloud, the edge (fog), and end devices. However, DDNN is designed for BranchyNet [32] and is hard to be extended to other types of DNN models. In [33], the authors presented a DNN as an encoding pipeline that encodes the feature space and transmits it to the clouds. It improves the energy efficiency and throughput of the model inference. Edgent [21] exploits two knobs: DNN partitioning and DNN right-sizing to find the optimal cutting point in a dynamic network environment. Hu et al. [22] proposed DADS, a partition scheme that optimally cut the DNN with DAG topology under different network conditions. However, it fails to reduce the overall delay because of the high time complexity of the algorithm. This method cannot guarantee real-time application. In reference [34], the authors studied the mobile Web AR scenario for edge-cloud collaboration and proposed a fine-grained adaptive DNN partition mechanism. In [35], the authors studied the edge-cloud inference of RNN models. It can decide whether to offload to clouds depending on network condition and input size. Wang et al. [36] proposed a task scheduling algorithm for tasks that need to be transferred to the cloud based on the catastrophic genetic algorithm (CGA) to satisfy the latency constraint. In [37], a novel DNN architecture was design for edge-cloud collaborative inference. However, this method is difficult to apply to currently running IoT applications. In [38], Auto-Split was proposed as an industry solution of DNN splitting for edge-cloud collaborative inference.

3. Problem Definition

First, we consider the chain topology DNN. For a given chain DNN model, we construct it as a chain $L = <V, E>$. Each vertex $v_i \in V$ corresponds to one layer in the model, and $<v_{i-1}, v_i> \in E$ corresponds to the two layers of model M with data transmission. $T^e(j, i)$ and $T^c(j, i)$ represent the delays from the j-th layer to the i-th layer at the edge and the cloud, respectively. For a single DNN, after using the edge-cloud collaborative inference strategy, if the cutting is performed on the i-th layer of the DNN, the first to i-th layers will be deployed on the edge, and the $i+1$th~Nth layer will be deployed in the cloud. The output d_i of vertex v_i will be transmitted to the cloud through the network. When the bandwidth is B, the total inference delay is $T^e(1, i) + d_i/B + T^c(i+1, N)$. The best cutting layer of a single DNN is the point where the total inference delay is minimized. The optimization objective of a single DNN is

$$T_{\min} = \operatorname{argmin}\left(T^e(1, i) + \frac{d_i}{B} + T^c(i+1, N)\right)(i = 0, 2, \cdots, N). \tag{1}$$

For chain DNN, we apply an iterative algorithm to enumerate the inference delay required by the DNN segmented in each layer and take the minimum value. In the above equation, $i = 0$ means that the entire DNN is deployed at the edge, and $i = N$ represents that the entire DNN is deployed in the cloud. When multiple edge devices generate tasks at the same time, at this time, each DNN has inference delay T_e, transmission delay T_t, and cloud inference delay T_c at the edge. Since the cloud is not always able to process the tasks sent from the edge in time, the waiting delay T_w of each task needs to be considered. At this time, the goal of optimization has changed from minimizing the delay of a single DNN to minimizing the overall delay of the system.

As shown in Figure 1, there are E edge servers and one cloud server in the multiedge scenario. Assume that E edge servers have a total of M DNN inference tasks, and all tasks use the same type of DNN for inference.

4. Method Design

In this section, we designed two scheduling strategies to solve the problem of partial migration of tasks in multiedge scenarios. The first strategy named Single Cutting treats each DNN as the same; this strategy is based on finding the optimal cutting position of a single DNN. The second strategy named Scheduling with Queuing takes other external factors into account; it adjusts the cutting position dynamic according to the network condition.

4.1. Single Cutting. For the scheduling strategy for each DNN inference task, we use the strategy that minimizes the delay of a single DNN for scheduling. Since we are using the same type of DNN, for each task, when the bandwidth is constant at B, $T_{\{\min\}}$ is the same. We use an iterative algorithm to find the edge processing time t_e and data transmission time at this time t_t, cloud processing time t_c, and optimal cutting point v_{best}.

We use FIFO to schedule all tasks. When there are tasks in task set S, the scheduler will not terminate. Each edge end e maintains a variable, which represents the earliest time at which the edge end can process the next task edgetime$_e$. In each poll, if there is still a task that has not been processed on the edge end e, we will update the current edgetime$_e$ to the maximum value between the arrival time of the current task and edgetime$_e$, and finally, add the current task processing time t_e.

After updating the time at which all edges can process the next task, we select the edge that can be processed

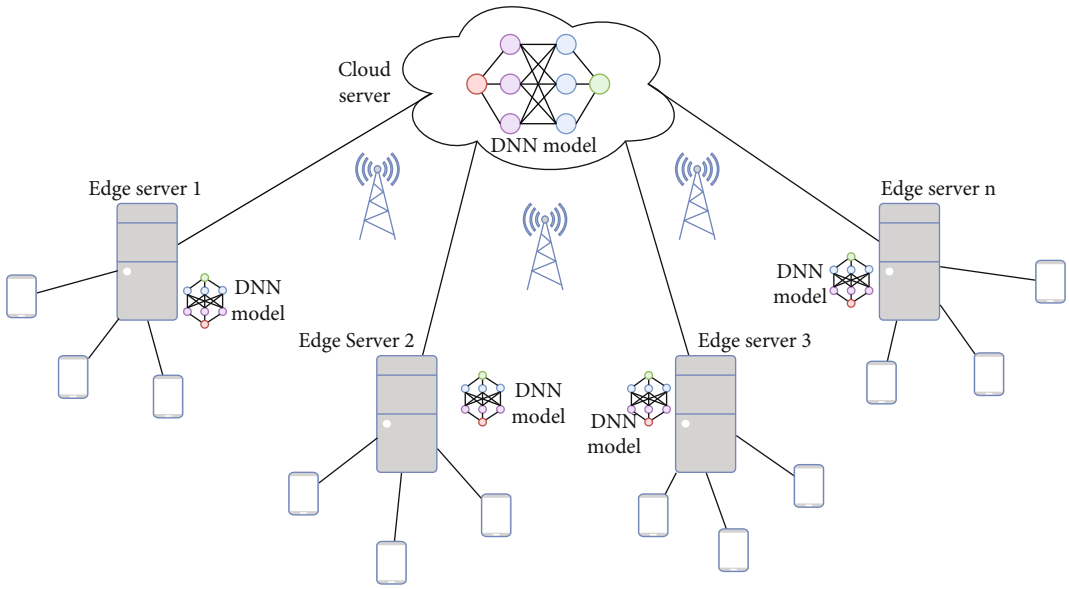


Figure 1: Model architecture in multiedge scenarios.

earliest among them. Next, we check whether the task queue C in the cloud is full. If the task queue is not full, we transfer the intermediate variables of the current task to the cloud task queue. Otherwise, the current task will enter a blocking state, that is, after waiting for the cloud to process the first task of the team, we will transfer the current task to the cloud for processing. After the cloud finishes processing a task, it updates the processing time of the cloud and wakes up the blocked task to start the next poll. The whole process is shown in Algorithm 1.

4.2. Scheduling with Queuing. Scheduling strategy 1 uses the optimal division of a single DNN for each task, but it is not necessarily the optimal solution globally, because no matter whether the cloud is currently congested or not, each DNN will choose the optimal cutting point under noncongested conditions for segmentation. It may leave the edge end in an idle state, and the cloud has been waiting for tasks, which will cause the overall time to become longer.

Suppose the current DNN inference task to be processed is task, and the start time at which the edge end can process it is t_{start}. The task is divided according to the optimal division of a single DNN, the time to reach the cloud is $t_{\text{start}} + t_e + t_t$, and the time when the previous task pretask is executed is t_{end}. If $t_{\text{start}} + t_e + t_t \geq t_{\text{end}}$, then there will be no waiting time for the DNN to reach the cloud. At this time, v_{best} is cut into the optimal solution of the task. If $t_{\text{start}} + t_e + t_t < t_{\text{end}}$, after the task arrives in the cloud, the waiting time is $t_{\text{wait}} = t_{\text{end}} - t_{\text{start}} - t_e - t_t$. At this time, a new cutting point can be selected after v_{best} to divide the task. In this case, you can select a new cutting point after v_{best} to divide the task. At the new cutting point, the edge end processing time is t'_e, the data transmission time is t'_t, the cloud processing time is t'_c, and the waiting time $t'_{\text{wait}} = t_{\text{end}} - t_{\text{start}} - t'_e - t'_t$. If $t'_{\text{wait}} < t_{\text{wait}}$, then for the task, the new cutting point is better than v_{best}, the waiting time will be less, and because the new cutting point is after the optimal cutting point, there will be fewer layers executed in the cloud.

However, when selecting a new cutting point, the execution time of the edge is $t'_e > t_e$, which causes the waiting time of the next DNN task to be processed too long. This may not be conducive to the overall delay reduction; we need to take into account the waiting delay of the next task in the edge device. Assume that the next task that needs to be inferred at the current edge is nexttask. At the original cutting point, we can get that the waiting time for nexttask is $t_{\text{start}} + t_e - \text{arrivetime}_{\text{nexttask}}$, then the overall waiting time of the system is

$$t_{\text{wait}} + t_{\text{start}} + t_e - \text{arrivetime}_{\text{nexttask}}. \tag{2}$$

$\text{arrivetime}_{\text{nexttask}}$ represents the arrival time of the next task. Similarly, the overall waiting delay of the system at the new cutting point is

$$t'_{\text{wait}} + t_{\text{start}} + t'_e - \text{arrivetime}_{\text{nexttask}}. \tag{3}$$

It is necessary to ensure that the overall delay of Equation (3) is less than Equation (2), that is, $t_{\text{wait}} - t'_{\text{wait}} > t'_e - t_e$. Based on the above analysis, we design scheduling strategy 2, and Algorithm 2 describes the process of scheduling strategy 2.

4.3. DAG-Type DNN Scheduling Method. In the above sections, we took the chain DNN as an example and proposed two scheduling algorithms. The first method is that for each DNN, we cut at the optimal cutting point of a single DNN and then schedule. The second method is to consider the waiting delay of the DNN in the cloud and find the cutting point that can make the overall delay smaller on the basis of the optimal cutting point of a single DNN to cut and then perform scheduling.

The cutting point of each DNN of Single Cutting is determined, so for DAG-type DNNs, we can modify the

Input: DNN topology L, layer N; bandwidth B; edge number E; task set S; task queue capacity C
Output: total time to execute all tasks cloudtime
$t_e, t_t, t_c, v_{\text{best}} \longleftarrow \text{BestPartition}(L, N, B)$
for $e = 1, 2, \cdots, E$ **do**
edgetime$[e] \longleftarrow 0$
end for
cloudtime $\longleftarrow 0$
$\text{cloud}_{\text{queue}} \longleftarrow \{\}$
needupdate $\longleftarrow$ true
while $S! = \varnothing$ **then**
  minedgetime $\longleftarrow +\infty$
  nowtask $\longleftarrow \varnothing$
  for $e \longleftarrow 1, 2, \cdots, E$ **do**
    task $\longleftarrow$ getLastArriveTask(S, e)
    if task! $= \varnothing$ and needupdate **then**
      edgetime$[e] \longleftarrow \max$ (edgetime$[e]$, task.arrivetime) $+ t_e$
    end if
    if minedgetime > edgetime$[e]$ **then**
      edgetime$[e] \longleftarrow$ minedgetime
      nowtask $\longleftarrow$ task
    end if
    if length$(\text{cloud}_{\text{queue}}) < C$ **then**
      add {nowtask, t_t, t_c} to $\text{cloud}_{\text{queue}}$
      delete nowtask from S
      needupdate $\longleftarrow$ true
    else then
      needupdate $\longleftarrow$ false
    end if
    process_update$(\text{cloud}_{\text{queue}}$.front, cloudtime$)$
  end for
end while
return cloudtime

Algorithm 1: Single Cutting.

BestPartition(L, N, B) function in Single Cutting to QDMP algorithm. The Scheduling with Queuing method needs to find the point that can make the waiting delay smaller by enumerating the cutting points to shorten the overall delay of the system. For the chain model, we can find the optimal cutting point by enumerating the cutting points sequentially. But for the DAG model, the optimal edge cut set often corresponds to a set of vertices and edges, and enumerating all points and edges is an NP-hard problem. We cannot find a better set of cut edges by enumerating vertices sequentially. Therefore, in this article, we only discuss Single Cutting on DAG-type DNNs.

5. Experimental Evaluation

5.1. Experimental Environment Settings. We use multiple edge devices and a single cloud for scheduling simulation. For the edge, we use 5 Raspberry Pi 3B platforms, which are equipped with a 4-core ARM Cortex-A53@1.2 GHz processor and 1 G RAM. For the cloud, we used a laboratory server equipped with an 8-core Intel core i7-9700 k@3.60 GHz processor and a NVIDIA RTX 2080Ti GPU. For the dataset, we use the self-acquired video dataset to evaluate our proposed scheduling algorithms. Each edge will sample the video frame, extract 5~15 frames of pictures from the video every second, and use the DNN model for inference. In the experiment, we considered six different DNN models, including three chain DNN models and three DAG-type DNN models. The three chain models are AlexNet, Tiny-YOLO, and DarkNet19, respectively. The three DAG models are AlexNet-Parallel, ResNet-18, and GoogLeNet. We evaluated the edge-only method, the cloud-only method, Single Cutting method, and Scheduling with Queuing method.

5.2. Inference Delay under 3G and 4G Networks. We first compare the inference delays of different methods in different network states. We use 3G and 4G as the default transmission technology, and the theoretical maximum uplink bandwidth is 1.1 Mbps and 5.85 Mbps, respectively. We use 5 Raspberry Pi 3B, each Raspberry Pi 3B will input a video stream, each edge end samples 30 frames of pictures in the video, and the cloud cache queue size is 20. In Table 1, we list the total inference delay of the system under different scheduling methods using different DNNs. For chain DNN, we use both two scheduling methods for scheduling. For DAG-type DNN, we use Single Cutting for scheduling.

Input: DNN topology L, layer N; bandwidth B; edge number E; task set S; task queue capacity C
Output: total time to execute all tasks cloudtime

$t_e, t_t, t_c, v_{\text{best}} \longleftarrow \text{BestPartition}(L, N, B)$
for $e = 1, 2, \cdots, E$ **do**
edgetime$[e] \longleftarrow 0$
end for
cloudtime $\longleftarrow 0$
$\text{cloud}_{\text{queue}} \longleftarrow \{\}$
needupdate $\longleftarrow$ true
while $S! = \varnothing$ **then**
  minedgetime $\longleftarrow +\infty$
  nowtask $\longleftarrow \varnothing$
    for $e \longleftarrow 1, 2, \cdots, E$ **do**
      task $\longleftarrow$ getLastArriveTask(S, e)
      next_task $\longleftarrow$ getSecondLastArriveTask(S, e)
      if task! $= \varnothing$ **then**
        edgetime$[e] \longleftarrow$ max (edgetime$[e]$, task.arrivetime)
    end if
    $t_{\text{wait}} \longleftarrow$ cloudtime $-$ edgetime$[e] - t_e - t_c$
    if $t_{\text{wait}} \le 0$ **then**
      $t_t' \longleftarrow t_t, t_e' \longleftarrow t_e, t_c' \longleftarrow t_c$
    else then
    for $i \longleftarrow v_{\text{best}} + 1, \cdots, N$ **do**
      $t'_{\text{wait}} \longleftarrow$ cloudtime $-$ edgetime$[e] - T^e(1, i) - d_i/B$
      if $t'_{\text{wait}} \ge 0$ and $t'_{\text{wait}} < t_{\text{wait}}$ and $t_{\text{wait}} - t'_{\text{wait}} > t_e' - t_e$ **then**
      $t_{\text{wait}} \longleftarrow t'_{\text{wait}}, t_e' \longleftarrow T^e(1, i), t_t' \longleftarrow d_i/B, t_c' \longleftarrow T^c(i + 1, N)$
      end if
    end for
  end if
  if needupdate **then**
    edgetime$[e] \longleftarrow$ edgetime$[e] + t_e'$
  end if
  if minedgetime > edgetime$[e]$ **then**
  edgetime$[e] \longleftarrow$ minedgetime
    nowtask $\longleftarrow$ task
  end if
  if length$(\text{cloud}_{\text{queue}}) < C$ **then**
    add {nowtask, t_t, t_c} to $\text{cloud}_{\text{queue}}$
    delete nowtask from S
    needupdate $\longleftarrow$ true
  else then
    needupdate $\longleftarrow$ false
  end if
    process_update($\text{cloud}_{\text{queue}}$.front(), cloudtime)
  end for
end while
return cloudtime

Algorithm 2: Scheduling with Queuing.

We further evaluated the speedup of different scheduling methods with the "edge-only" delay as a baseline. S-C and S-Q represent the inference delay of Single Cutting strategy and Scheduling with Queuing, respectively, and the speedup is defined as

$$\kappa = \frac{L_{\text{EdgeOnly}}}{L_{\text{ALGO}}}, \tag{4}$$

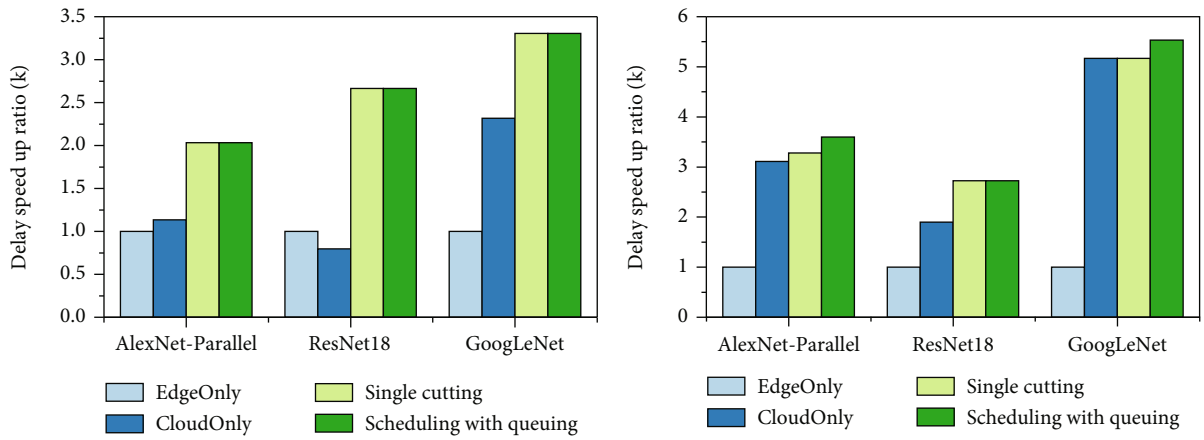
where L_{EdgeOnly} represents the inference delay of only the edge and L_{ALGO} represents the inference delay of all comparison methods. "Edge-only" is used as the baseline, and its speedup ratio is 1. The edge-only method executes the whole task on edge devices while the cloud-only method executes the whole task on the cloud server.

Figure 2 compares the chain DNN. Figure 2(a) shows the latency acceleration ratios of the four methods in the case of 3G. Both Single Cutting and Scheduling with

Table 1: The total inference delay of several typical DNN models under different scheduling methods (seconds).

DNN	3G				4G			
	C-O[1]	E-O[2]	S-C[3]	S-Q[4]	C-O[1]	E-O[2]	S-C[3]	S-Q[4]
AlexNet	78.3	88.9	43.7	43.7	28.6	88.9	27.1	24.7
Tiny-YOLO	200.8	159.7	59.9	59.9	84.2	159.7	58.6	58.6
DarkNet19	77.8	180.4	54.6	54.6	34.9	180.4	34.9	32.6
AlexNet-Parallel	70.2	44.9	40.6	—	27.9	44.9	27.9	—
ResNet18	84.3	81.7	32.0	—	34.4	81.7	25.5	—
GoogLeNet	91.2	121.1	63.3	—	43.2	121.1	43.2	—

[1]Cloud-only, [2]edge-only, [3]Single Cutting, and [4]Scheduling with Queuing.



(a) 3G Network (b) 4G Network

Figure 2: Inference delay acceleration ratio of 4 methods: Single Cutting, Scheduling with Queuing, cloud-only, and edge-only.

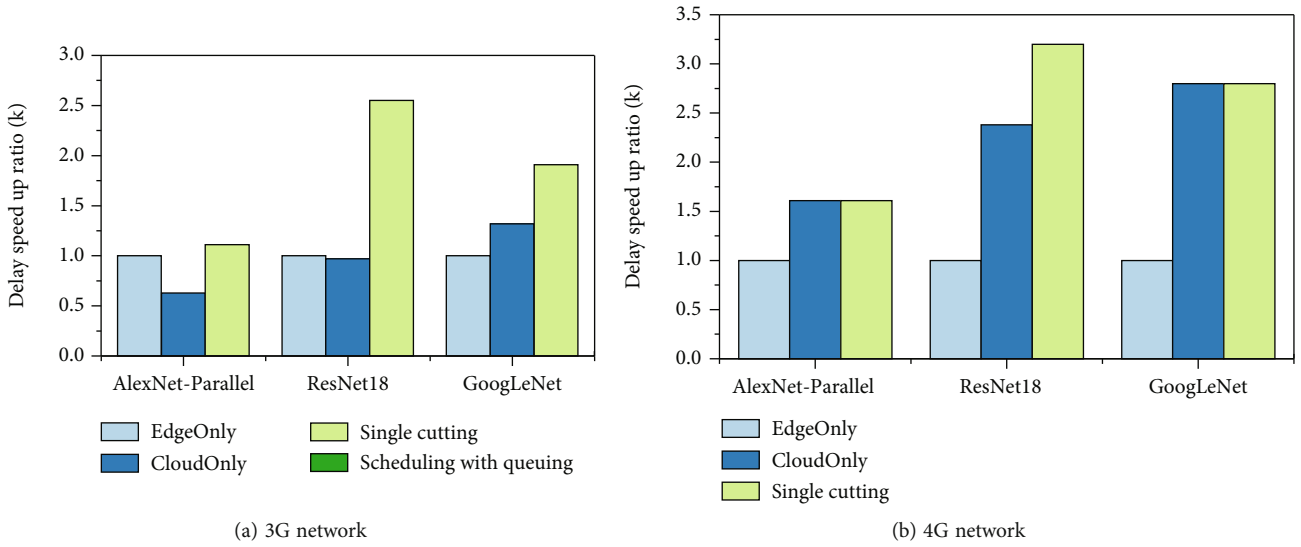
Queuing methods surpass the edge-only and cloud-only methods. In the case of 3G, the efficiency of Single Cutting and Scheduling with Queuing is the same. Compared with the edge-only and cloud-only methods, they achieve a delay acceleration of 2.03 to 3.30 times and 2.22 to 6.48 times, respectively. Figure 2(b) shows the delay acceleration ratios of the four methods in the case of 4G. Compared with only the edge end and only the cloud, the Single Cutting and the Scheduling with Queuing achieve 1.9~5.48 and 1.05~1.57 times acceleration, respectively. In both AlexNet and DarkNet19 models, Scheduling with Queuing achieves the optimal delay speedup ratio. On Tiny-YOLO, Single Cutting and Scheduling with Queuing have the same efficiency.

Figure 3 compares the DAG-type DNN. Figure 3(a) shows the latency acceleration ratios of the three methods in the case of 3G. The cloud-only method performs the worst due to network bandwidth limitations. Compared with the edge-only and cloud-only methods, the Single Cutting achieves 1.11~2.72 times and 2.07~4.12 times delay acceleration, respectively. Figure 3(b) shows the latency acceleration ratios of the three methods in the case of 4G. The cloud-only method surpasses the edge-only method on the three models. On AlexNet-Parallel and GoogLeNet, the efficiency of Single Cutting is the same as that of the cloud. On ResNet18, the efficiency of Single Cutting is 1.34 times higher than that of the cloud alone. Compared with the edge-only method, the Single Cutting achieves a speedup of 1.11~2.72 times.

5.3. The Influence of the Number of Edge Devices on the Inference Delay. In this section, we compare the performance of different methods with different numbers of edge ends. In the same way, we use "edge-only" as the baseline to evaluate the speedup ratio (k) of different methods. We use AlexNet and ResNet18 to verify the effectiveness of our scheduling algorithm under different edge numbers.

As shown in Figure 4, we deployed AlexNet on the edge and compared the speedup ratios of the four algorithms on the chain DNN in the case of 1 to 5 edge ends. Single Cutting and Scheduling with Queuing are better than the edge-only and cloud-only methods under different numbers of edge terminals. Compared with the edge-only method, the Single Cutting can achieve a delay acceleration of 2.88 to 3.27 times, and the Scheduling with Queuing can achieve a delay acceleration of 3.12 to 3.75 times.

As shown in Figure 5, we deploy ResNet18 on the edge and compare the speedup ratios of the three algorithms on DAG-type DNNs in the case of 1 to 5 edge ends. Compared with the edge-only method, scheduling Algorithm 1 can achieve a delay acceleration of 2.94 to 3.20 times. Compared with the edge-only method, scheduling Algorithm 1 can achieve a delay acceleration of 2.94~3.20 times. Compared with the cloud-only method,



(a) 3G network (b) 4G network

Figure 3: Inference delay acceleration ratio of 3 methods: Single Cutting, cloud-only, and edge-only.

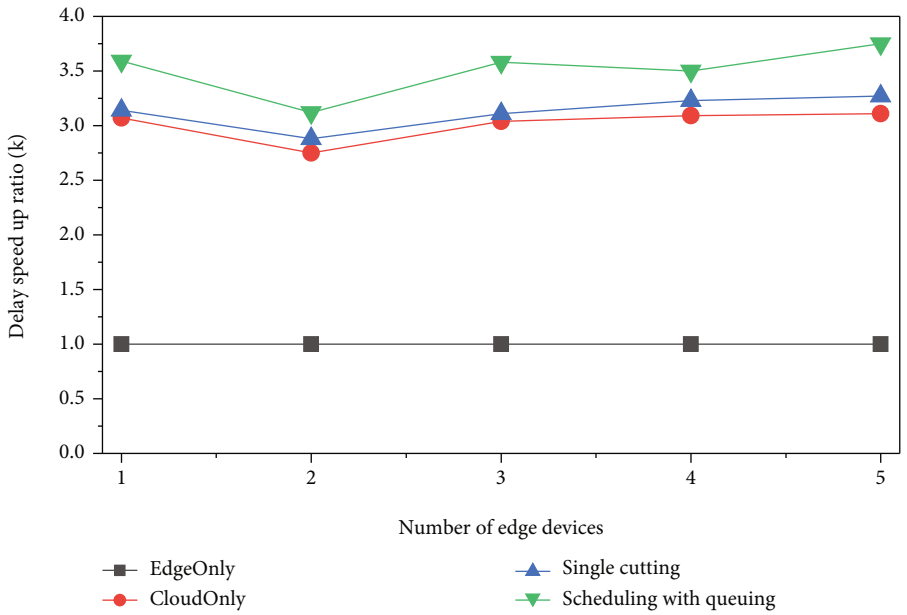


Figure 4: The delay acceleration ratio of AlexNet under different methods under different number of edge devices.

scheduling Algorithm 1 can achieve a delay acceleration of 1.25~1.43 times.

5.4. The Influence of Network Bandwidth on Inference Delay. We compared the impact of network bandwidth on the performance of different methods. We use "edge-only" as the baseline to evaluate the speedup (k) of different methods. During the experiment, we accelerated the network bandwidth from 0 Mbps to 12 Mbps. Similarly, we used AlexNet and ResNet18 to verify the influence of network bandwidth on our scheduling algorithm.

As shown in Figure 6, we deployed AlexNet at the edge and experimented with different methods on the impact of AlexNet's inference delay under different network conditions. When the network condition is 0 Mbps, the cloud-only method performs the worst, and the efficiency of the two scheduling algorithms and the edge-only method is the same. When the network condition is 1 Mbps, Single Cutting and Scheduling with Queuing have the same efficiency, which is better than the two baseline methods. When the network condition is between 2 Mbps and 7 Mbps, the efficiency of Scheduling with Queuing is higher than that of Single Cutting and the edge-only method. When the network condition is greater than 7 Mbps, Single Cutting, Scheduling with Queuing, and the edge-only method have the same efficiency.

As shown in Figure 7, we deployed ResNet18 at the edge and experimented with different methods on the inference

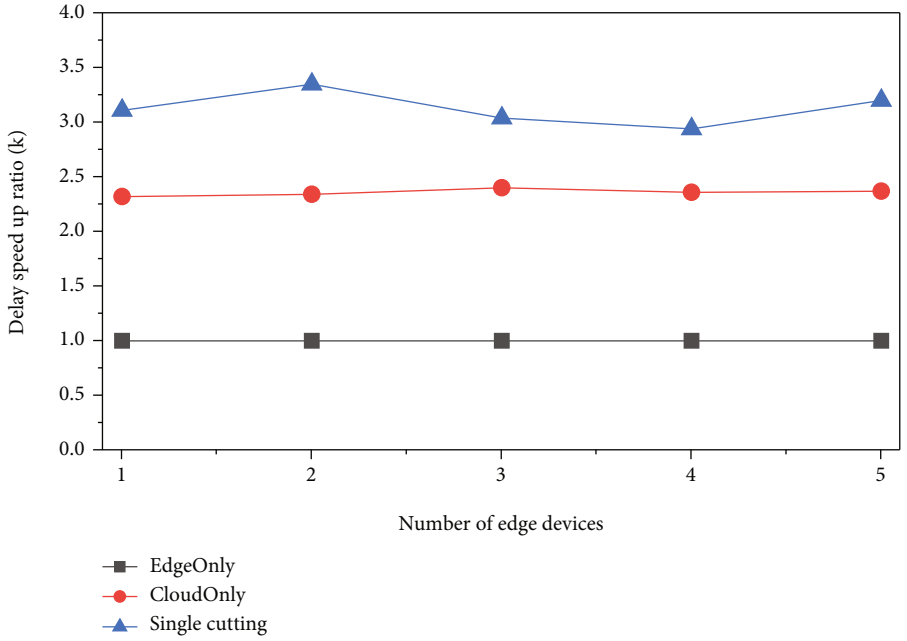


Figure 5: Delay acceleration ratio of ResNet under different methods under different numbers of edge ends.

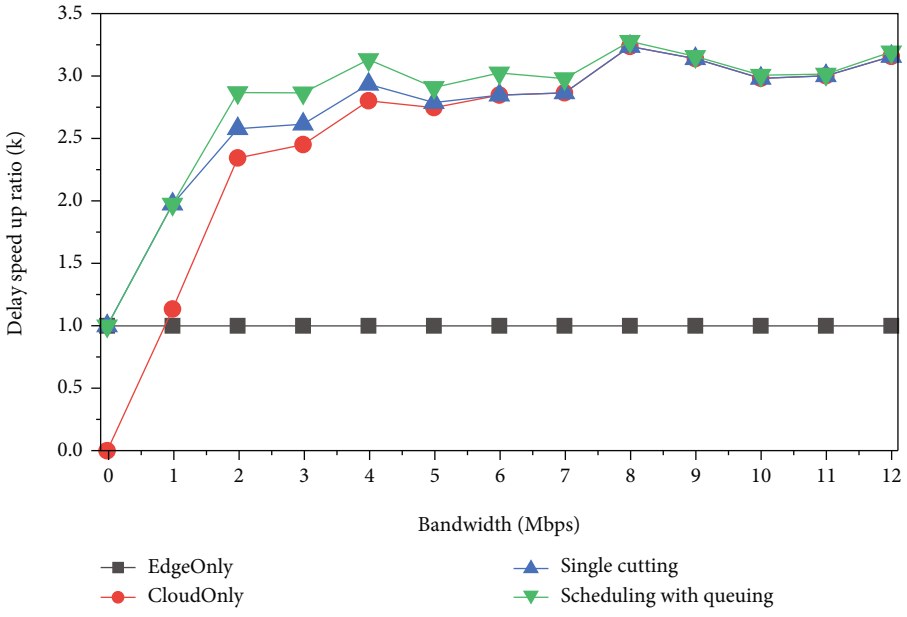


Figure 6: The impact of different methods of bandwidth changes on AlexNet inference delay.

delay of ResNet18 under different network conditions. When the network condition is 0 Mbps, the cloud-only method performs the worst, and the scheduling algorithm is the same as the edge-only method. When the network conditions are between 1 Mbps and 8 Mbps, scheduling Algorithm 1 performs optimally, at most 2.85 times faster than the edge-only method and at most 1.93 times faster than the cloud-only method. When the network speed is greater than 8 Mbps, scheduling Algorithm 1 has the same efficiency as the edge-only method.

6. Discussion

In Section 5, we implement the experiments using five Raspberry Pi platforms to mimic edge devices. According to the experimental results, the proposed scheduling algorithm can handle the circumstances that have more than 5 edge devices. With the increase of the number of edge devices, the number of tasks will be enormous, and the network resources are constrained; the first scheduling algorithm, i.e., Single Cutting, may not be able to find the optimal

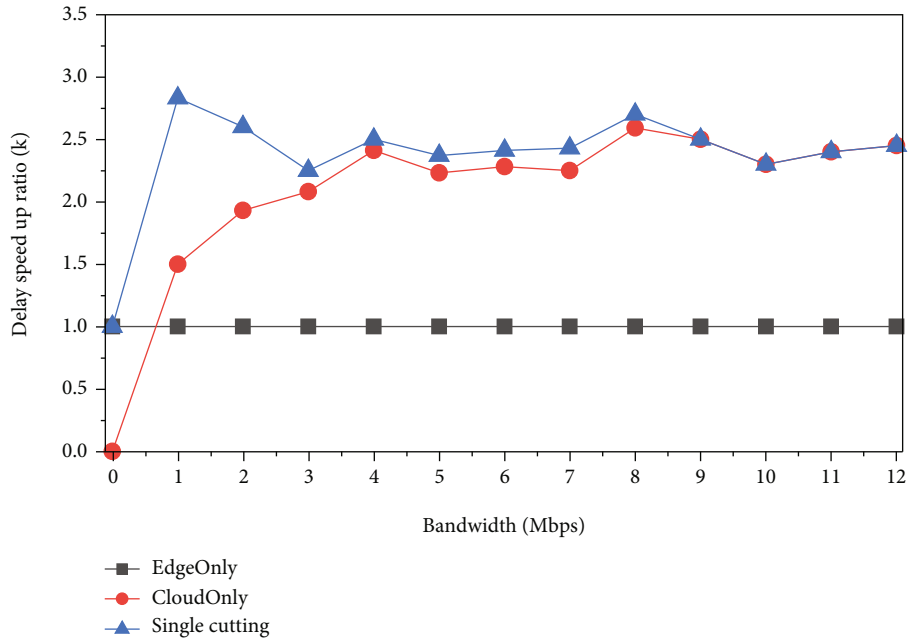


Figure 7: The impact of different methods of bandwidth changes on ResNet inference delay.

cutting point and fail to accelerate the inference, while the second scheduling algorithm, considering the network delay and queuing delay, can still find good cutting point to reduce the overall delay. The impact of cloud server is reflected in the queuing delay in the second scheduling algorithm; the larger the capacity of the cloud server, the less the queuing delay when scheduling. Moreover, different capacity of edge devices and cloud servers will cause different DNN model partitioning in proposed algorithms while the overall delay keeps low.

7. Conclusion

In this paper, we propose two DNN scheduling algorithms for edge-cloud collaborative inference systems. The difference from previous work is that, in the case of multiple edges, we considered partial migration of tasks instead of whole migration. The first scheduling algorithm performs scheduling based on the optimal decision of a single DNN, and each task performs task migration at the same split point. The algorithm combined with the QDMP algorithm can be applied to the scheduling of chain DNN and DAG-type DNN and has a wide range of applicability. The second scheduling algorithm can search for a partition point that can make the overall delay smaller for scheduling based on factors such as the waiting time for tasks in the cloud and the execution interval between tasks. This scheduling algorithm can be used for chain DNN scheduling.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Shigeng Zhang and Yue Zhang are the corresponding authors of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61772559, 61901529, 61902434 and the Natural Science Foundation of Hunan under Grant 2020JJ5776 and 2019JJ50826.

References

[1] K. Zhang, Y. Zhu, S. Maharjan, and Y. Zhang, "Edge intelligence and blockchain empowered 5g beyond for the industrial internet of things," *IEEE Network*, vol. 33, no. 5, pp. 12–19, 2019.

[2] S. Zhang, X. Liu, Y. Liu, B. Ding, S. Guo, and J. Wang, "Accurate respiration monitoring for mobile users with commercial rfid devices," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 513–525, 2020.

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

[4] S. Pearson, "Privacy, security and trust in cloud computing," in *Privacy and Security for Cloud Computing*, pp. 3–42, Springer, 2013.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] Y. Kang, J. Hauswald, C. Gao et al., "Neurosurgeon," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[7] H. Li, K. Ota, and M. Dong, "Learning iot in edge: deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

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

[9] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 784–800, Munich, Germany, 2018.

[10] J. Dean, G. S. Corrado, R. Monga et al., "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems*, pp. 1232–1240, Lake Tahoe, NV, USA, December 2012.

[11] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[12] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," in *6th International Conference on Learning Representations, ICLR*, Vancouver, BC, Canada, April 2018.

[13] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[14] W. Park, D. Kim, L. Yan, and M. Cho, "Relational knowledge distillation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3967–3976, Long Beach, CA, USA, June 2019.

[15] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017, http://arxiv.org/abs/1710.09282.

[16] S. Han, X. Liu, H. Mao et al., "EIE: efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[17] W. Jiang, Z. He, S. Zhang et al., "Microrec: accelerating deep recommendation systems to microseconds by hardware and data structure solutions," 2020, http://arxiv.org/abs/2010.05894.

[18] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating cnn inference on asics: a survey," *Journal of Systems Architecture*, vol. 113, article 101887, 2021.

[19] K. Guo, L. Sui, J. Qiu et al., "From model to fpga: software-hardware co-design for efficient neural network acceleration," in *2016 IEEE Hot Chips 28 Symposium (HCS)*, pp. 1–27, Cupertino, CA, USA, August 2016.

[20] S. Zhang, Y. Li, X. Liu et al., "Towards real-time cooperative deep inference over the cloud and edge end devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–24, 2020.

[21] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: on-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[22] H. Chuang, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1423–1431, Paris, France, April 2019.

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

[24] X. Liu, J. Yin, S. Zhang, B. Xiao, and B. Ou, "Time-efficient target tags information collection in large-scale RFID systems," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 2891–2905, 2021.

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

[26] X. Liu, J. Yin, J. Liu, S. Zhang, and B. Xiao, "Time efficient tag searching in large-scale RFID systems: a compact exclusive validation method," *IEEE Transactions on Mobile Computing*, vol. 1, p. 1, 2020.

[27] X. Liu, Q. Yang, J. Luo, B. Ding, and S. Zhang, "An energy-aware offloading framework for edge-augmented mobile rfid systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 3994–4004, 2018.

[28] R. Gao, H. Yang, S. Huang et al., "Pripro: towards effective privacy protection on edge-cloud system running dnn inference," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 334–343, Melbourne, Australia, May 2021.

[29] Y. Dong, P. Zhao, H. Yu, C. Zhao, and S. Yang, "CDC: classification driven compression for bandwidth efficient edge-cloud collaborative deep learning," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, pp. 3378–3384, 2020.

[30] A. G. Howard, M. Zhu, B. Chen et al., *Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017, http://arxiv.org/abs/1704.04861.

[31] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, Atlanta, GA, USA, June 2017.

[32] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, Cancun, Mexico, December 2016.

[33] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, Auckland, New Zealand, November 2018.

[34] P. Ren, X. Qiao, Y. Huang, L. Liu, S. Dustdar, and J. Chen, "Edge-assisted distributed dnn collaborative computing approach for mobile web augmented reality in 5g networks," *IEEE Network*, vol. 34, no. 2, pp. 254–261, 2020.

[35] D. J. Pagliari, R. Chiaro, Y. Chen, E. Macii, and M. Poncino, "Optimal input-dependent edge-cloud partitioning for rnn inference," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 442–445, Genoa, Italy, November 2019.

[36] S. Wang, Y. Li, S. Pang, Q. Lu, S. Wang, and J. Zhao, "A task scheduling strategy in edge-cloud collaborative scenario based

on deadline," *Scientific Programming*, vol. 2020, Article ID 3967847, 9 pages, 2020.

[37] H. Zhou, W. Zhang, C. Wang, X. Ma, and H. Yu, "Bbnet: a novel convolutional neural network structure in edge-cloud collaborative inference," *Sensors*, vol. 21, no. 13, p. 4494, 2021.

[38] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: a general framework of collaborative edge-cloud ai," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2543–2553, Singapore, August 2021.