Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 9985006, 12 pages
https://doi.org/10.1155/2021/9985006

Research Article

Conflict-Resilient Incremental Offloading of Deep Neural Networks to the Edge of Smart Environment

Zhongmin Chen,[1,2,3] **Zhiwei Xu**,[1,2,4] **Jianxiong Wan**,[1,3] **Jie Tian**,[5] **Limin Liu**,[1] **and Yujun Zhang**[2]

[1]*College of Data Science and Application, Inner Mongolia University of Technology, Hohhot 100080, China*
[2]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
[3]*Inner Mongolia Autonomous Region Engineering & Technology Research Center of Big Data Based Software Service, Hohhot 100080, China*
[4]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[5]*Department of Computer Science, New Jersey Institute of Technology, 323 Dr Martin Luther King Jr Blvd, Newark, NJ 07102, USA*

Correspondence should be addressed to Zhiwei Xu; xuzhiwei2001@ict.ac.cn

Received 30 March 2021; Revised 8 May 2021; Accepted 25 May 2021; Published 7 June 2021

Academic Editor: Wenyu Zhang

Novel smart environments, such as smart home, smart city, and intelligent transportation, are driving increasing interest in deploying deep neural networks (DNN) in edge devices. Unfortunately, deploying DNN at resource-constrained edge devices poses a huge challenge. These workloads are computationally intensive. Moreover, the edge server-based approach may be affected by incidental factors, such as network jitters and conflicts, when multiple tasks are offloaded to the same device. A rational workload scheduling for smart environments is highly desired. In this work, we propose a Conflict-resilient Incremental Offloading of Deep Neural Networks at Edge (CIODE) for improving the efficiency of DNN inference in the edge smart environment. CIODE divides the DNN model into several partitions by layer and incrementally uploads them to local edge nodes. We design a waiting lock-based scheduling paradigm to choose edge devices for DNN layers to be offloaded. In detail, an advanced lock mechanism is proposed to handle concurrency conflicts. Real-world testbed-based experiments demonstrate that, compared with other state-of-the-art baselines, CIODE outperforms the DNN inference performance of these popular baselines by 20% to 70% and significantly improves the robustness under the insight of neighboring collaboration.

1. Introduction

In recent years, deep neural networks (DNN) have been widely used in edge environments. From smart home [1] and smart city [2] to automatic driving [3], everyday objects around us are becoming "smarter," by integrating terminals related to daily life and realizing a smart, safe, convenient, artistic, and energy-saving environment based on DNNs. According to Internet Data Center (IDC) [4], there will be 41.6 billion edge devices deployed in smart environments by 2025, and their computing capability will gradually increase. Such a large number of smart terminals will bring about many challenges. As a new paradigm to deal with these challenges, edge intelligence [5] is widely regarded as a key technology to realize various ideas of the next generation of the Internet (e.g., Tactile Internet) [6]. It collects computing resources and storage resources distributed in smart environments to perform computing-intensive or delay-critical computing tasks on edge devices. However, DNN applications are too computationally intensive to execute on a resource-constrained edge node.

To solve the above problem, a traditional strategy is to upload computing tasks to a central cloud server [7] and run DNN computing tasks with the powerful hardware platform of the server. It requires the DNN model to be preinstalled on the target server. Similarly, in edge environments, edge nodes can send their DNN computing tasks to nearby general edge servers [8]. However, the hardware capacity of edge servers is much smaller than that of centralized servers, and it is impossible for edge servers to store a large number of DNN models [9].

In recent years, studies have proposed using edge nodes and edge servers to collaboratively execute DNN. A popular method is to upload DNN models from edge nodes to edge servers on demand for DNN computation. Neurosurgeon [10] is a work that uses the DNN partition scheme to collaboratively execute DNN. Neurosurgeon divides the DNN model into the front part and the rear part. The edge node executes the front part, and the edge server runs the rear part using the delivered matrix and sends the output matrix back to the edge node. Neurosurgeon proved that the collaborative execution of DNN is effective, but it is based on an edge-cloud server with preinstalled DNN models, which does not meet the vision of uploading DNN models on demand. IONN [11] and its follow-up work, Enhanced IONN [12], used the DNN partition scheme to collaboratively execute DNN. They divide the DNN model into several partitions and upload them to the edge server in order. The edge server builds the DNN model incrementally, so it is allowed to start part of the DNN execution before uploading is completed. However, they have the problem of running gaps due to large DNN partitions. Moreover, they rely on a single edge server, which is easily affected by network jitters and other unstable factors. It will cause congestion and increased time cost and become unstable. Now, the most promising method is to completely push computing tasks to the edge of the Internet, even without the assistance of edge servers.

Collaborating multiple edge nodes to perform DNN tasks is an effective solution. The advantage of this approach is to reduce dependence on the cloud server and improve stability. Unfortunately, running DNN applications at edge faces its own challenges. Edge devices cannot meet the computing requirements of most DNN applications. For this reason, a lot of researches investigated effective methods of deploying DNN to the edge. NestDNN [13] and the work in [14] can efficiently merge the limited resources in the edge cluster to maximize the performance of all parallel applications. DeepThings [15] minimizes memory usage and communication costs by fusing early convolutional layers and parallelizing these layers in multiple devices. Despite all these efforts, improving the efficiency of DNN applications without reducing the model accuracy and processing the DNN queries in real time present ongoing challenges to us.

In this work, we explore parallel execution of DNN inference across multiple edge nodes. This does not require additional model adjustment or hardware deployment. A typical application we envisioned for our work is the local smart home. In this scenario, the computing tasks of the smart device can be offloaded to the smart device inside the house. For example, the smart door lock can request the smart refrigerator and the smart air conditioner to perform facial unlocking tasks. Furthermore, this work may be extended to other applications, including smart cities and Internet of vehicles [16].

Specifically, we propose the Conflict-resilient Incremental Offloading of Deep Neural Networks at Edge (CIODE) for improving the efficiency of DNN inference at the edge smart environment. We design an algorithm to determine the collaborative computing strategy. We also effectively deal with concurrency conflict exceptions to ensure the execution efficiency of DNN applications. Experimental evaluation shows that CIODE can dynamically select collaboration targets among trusted edge clusters. By collaborating with multiple edge nodes for DNN computing, it effectively avoids the influence of unstable factors and ensures the execution efficiency of DNN applications.

The primary contributions of this paper are listed as follows:

(1) In order to completely push the computing tasks of DNN applications to the edge of the network, after studying the challenges that need to be overcome to execute DNN applications in collaboration with multiple edge nodes, we found some problems that may delay the execution of DNN applications, such as concurrency conflict exceptions, network jitters, and deadlocks.

(2) In order to improve the performance of DNN applications and solve the above problems, we propose the CIODE. CIODE can dynamically select collaborative targets among trusted edge clusters and collaborate with multiple targets to execute DNN applications. We also design an advanced lock mechanism for CIODE to handle concurrent conflict exceptions. In this way, CIODE can improve the query performance while enhancing the robustness of DNN applications.

(3) We implement CIODE as well as other baselines in a real edge computing environment consisting of edge devices with different hardware configurations and support image recognition with DNN models. Experimental results show that CIODE can improve the inference efficiency of DNN models, handle concurrency conflict exceptions, and significantly improve the robustness of the system while ensuring performance.

The rest of this paper is organized as follows. Section 2 details the related work. Section 3 establishes a graph model to clarify optimization goals. Section 4 introduces our method and its algorithm in detail. Section 5 conducts a comprehensive evaluation. Section 6 concludes the paper.

2. Related Work

The technologies that support DNN applications fall into two categories: cloud-based approaches and edge-only approaches. Cloud-based approaches offload DNNs to cloud

servers. Edge-only approaches execute DNNs only on a single edge device or a small Internet of Things (IoT) cluster.

2.1. Cloud-Based Approaches. Cloud-based approaches [10–12, 17–24] offload all (i.e., cloud-only) or part (i.e., edge-cloud collaboration) of the DNN computation to the cloud server. Neurosurgeon [10] dynamically divides the DNN model into the front part and the rear part. The edge node executes the front part, and the edge server runs the rear part using the delivered matrix and sends the output matrix back to the edge node. However, Neurosurgeon is based on a cloud server with preinstalled DNN models, and its partition algorithm does not consider the upload overhead. In case the model is not preinstalled when offloading remotely, IONN [11] and its follow-up work, Enhanced IONN [12], divide the DNN model into several partitions and upload them to the edge server in order, which builds the DNN model incrementally. In [17], some DNN computing tasks are offloaded to the cloud to minimize delay and resource consumption. Hu C et al. [19] designed an adaptive DNN splitting algorithm, which can find the optimal splitting strategy under dynamic and time-varying network conditions. Ren et al. [20] further transformed the original joint communication and computing resource allocation problem into an equivalent convex optimization problem and used convex optimization theory to obtain a closed-form computing resource allocation strategy. Gazzaz et al. [21] proposed a collaborative system solution in which the videos uploaded on edge nodes can be tagged and collaboratively analyzed. Wang et al. [22] described the computation offloading problem in the multiaccess edge environment as a deep sequential model based on reinforcement learning, which can automatically discover the common patterns behind various applications. In order to minimize the execution delay between the client and the edge server, PerDNN [23] uses the server's GPU statistics to split DNN models. Xu et al. [24] focus on offloading DNNs based on one or more cloudlets to minimize the energy consumption or maximize the inference requests.

However, the above cloud-based approaches all depend on unpredictable cloud availability and unstable communication. In addition, the above approaches also have the risk of exposing privacy, and the information security cannot be guaranteed. On the contrary, our work can effectively avoid the above problems such as running gaps, network jitters, and information security by layering partitions and collaborate with multiple edge nodes for DNN execution.

2.2. Edge-Only Approaches. Edge-only approaches [13–15, 25–32] execute DNNs only on a single edge device or a small IoT cluster. NestDNN [13] can efficiently use the limited resources in edge clusters and maximize the performance of all parallel applications. The work in [14] summarizes the computing resource of several edge nodes to achieve efficient, dynamic, and real-time identification. DeepThings [15] minimizes memory usage and communication costs by fusing early convolutional layers and parallelizing these layers in multiple devices. In [25], the authors enable the utilization of the aggregated computing capability of several IoT devices by creating a local collaborative network for a subset of DNNs. In this approach, IoT devices cooperate to conduct single-batch inferencing in real time. Modnn [27] uses multiple edge devices to perform DNN inference and accelerate task inference by reducing the computational cost and memory usage of a single device. Edge accelerators [30] use dynamic methods to determine the best split across layers. When model compression or model splitting is used, they provide delay and bandwidth advantages for cross-layer and cross-layer split processing. Reference [31] proposes a technique to divide a DNN into multiple partitions that can be processed locally by end devices or offloaded to one or multiple powerful nodes, such as in fog networks.

Our work falls into the category of collaborative DNN inference in multiple edge nodes. This method can effectively protect the privacy [32] and is not easily affected by unstable factors. In this way, by reasonably selecting collaboration targets among trusted edge devices to execute DNN applications, it improves the efficiency of DNN applications while protecting privacy. At the same time, it also effectively avoids the influence of the network jitter.

3. Problem Definition

The neural network execution graph can intuitively represent the process and the overhead of collaborative execution. There are some edge nodes ($D1$, $D2\ldots$, Dk) in Figure 1. They can request each other for collaborative computing. For the convenience of description, we take D1 as the requester and D2 to Dk as the set of trusted devices of D1. This simple neural network model has three layers (A, B, and C), and each layer has two nodes corresponding to each trusted device. In this figure, the leftmost nodes belong to the requester, and nodes on the right correspond to each trusted device. The path between nodes belonging to the requester means local execution. The path between the node belonging to the requester and the node belonging to the trusted device means the transmission of input or output data. The path between nodes belonging to the trusted device in the same layer indicates that it is executed on the trusted device.

In addition, we add weight to each path to represent the cost. Some paths have zero weight (the dashed path in Figure 1) because there is no computation or transmission overhead involved. When the DNN application is installed on the edge node, the device will run its DNN model and record the execution time of each DNN layer. However, the collaboration target cannot know which DNN model will be executed. To predict the execution time, it runs different DNN models with different layer parameters during deployment to perform linear regression on the execution data of DNN layers, as [10] does.

Also, the layer upload time and data transmission time can be calculated by dividing the size of layer or data by the current network speed. Before offloading, the current network speed is measured in real time and used to accurately estimate the transmission delay. This can be adapted to

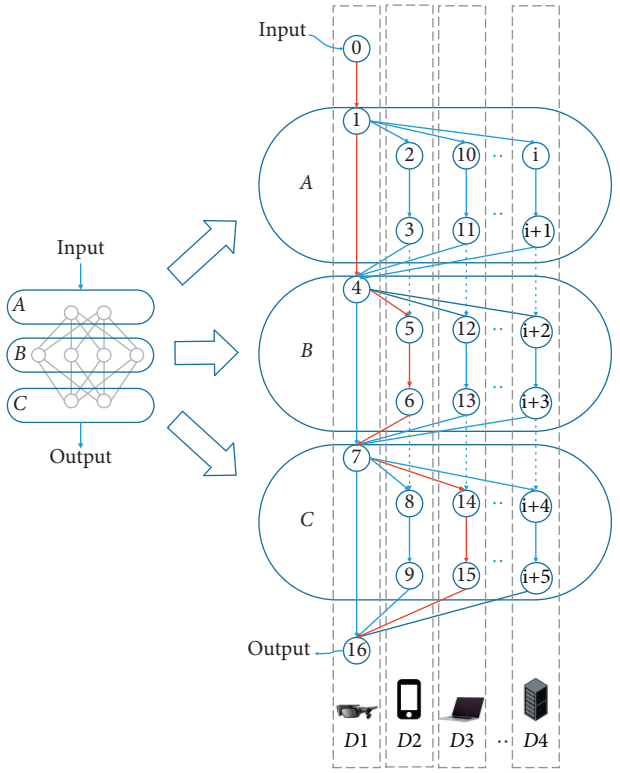

Figure 1: Neural network execution graph.

various network situations. The current network speed is calculated by the following formula:

$$N = \frac{\text{delivered}}{\text{interval}} \cdot \frac{1}{8}, \tag{1}$$

where N is the current network speed, delivered is the number of responding data packets, and interval is the time taken to respond to the above data packet. Among them, (delivered/interval) calculates the real-time bandwidth.

The transmission time includes the upload time of layers and the transmission time of the input or output data, which are calculated by formulas (2)–(4):

$$U_{\text{layer}} = \frac{S_{\text{layer}}}{N}, \tag{2}$$

where U_{layer} estimates the layer upload time and S_{layer} is the size of the layer.

$$T_{\text{input}} = \frac{S_{\text{input}}}{N}, \tag{3}$$

where T_{input} estimates the data transmission time of input and S_{input} is the size of the input data.

$$T_{\text{output}} = \frac{S_{\text{output}}}{N}, \tag{4}$$

where T_{output} estimates the data transmission time of output and S_{output} is the size of the output data.

Therefore, the overall delay of offloading layer i to the target device can be calculated by the following formula:

$$D_{\text{offload}}(i) = T_{\text{input}}(i) + E_{\text{target}}(i) + U_{\text{target}}(i) + T_{\text{output}}(i), \tag{5}$$

where $D_{\text{offload}}(i)$ estimates the overall delay of offloading and running layer i. $E_{\text{target}}(i)$ is the predicted execution time of layer i on target edge node. $U_{\text{target}}(i)$ is the upload time of layer i on target edge node.

The overall delay of offloading DNN models according to the offloading strategy can be calculated by the following formula:

$$D_{\text{total}} = \sum_{i \in L} E_{\text{local}}(i) + \sum_{j \in O} D_{\text{offload}}(j), \tag{6}$$

where D_{total} estimates the overall of offloading DNN models. L is the set of layers executed locally. O is the set of offloaded layers. $E_{\text{local}}(i)$ is the predicted execution time of layer i. $D_{\text{offload}}(j)$ is the overall delay of offloading layer j to another device.

There is a special case where our neural network execution graph does not work. There are some parallel layers in the DNN structure; they share the output data from the same layer and then pass their output data to another layer. To solve this problem, first, we find the *dominators* of the output layer. These *dominators* are the layers that must be included in the path from input to output [33]. In addition, we regard layers between two neighboring *dominators* and the latter *dominator* as just one layer. Then, the weight of the path in this layer is the sum of the weights of the above layers.

The shortest path on the neural network execution graph includes which layers should be uploaded to minimize execution time, and the direction of the path can indicate the execution flow. For example, the requester (D1) inputs DNN query to start execution. Layer A is executed locally on D1, and the execution cost will be represented by the weight of path 1 ⟶ 4. Layer B needs to be uploaded to D2 for collaborative execution; then the execution cost will be represented by the sum of the weights of paths 4 ⟶ 5, 5 ⟶ 6, and 6 ⟶ 7. The following layer C needs to be uploaded to D3 for collaborative execution; then the execution cost will be represented by the sum of the weights of paths 7 ⟶ 14, 14 ⟶ 15, and 15 ⟶ 16. In this way, the path from input node (node 0) to output node (node 16) represents the execution flow of executing the entire DNN task, that is, 0 ⟶ 1 ⟶ 4 ⟶ 5 ⟶ 6 ⟶ 7 ⟶ 14 ⟶ 15 ⟶ 16 (the red path in Figure 1). The corresponding execution process is as follows: first, D1 executes layer A locally and then uploads layer B to D2 for execution and returns output data; finally, it uploads layer C to D3 for execution and returns the result. Now, the sum of the weights of the path from node 0 to node 16 represents the overall delay of executing the DNN task.

Therefore, the problem we are studying is how to determine DNN layers that need to be uploaded, the upload order, and collaboration targets, so as to improve inference efficiency to achieve the lowest execution delay.

4. CIODE: Conflict-Resilient Incremental Offloading of Deep Neural Networks at Edge

4.1. Overall Design. In this section, based on the neural network execution graph (in Section 3), we propose the CIODE. It can solve the optimization problem of how to determine the DNN layers that need to be uploaded, the upload order, and the collaborative devices during DNN collaborative computation. In addition, we design the waiting lock and the invalid lock to solve concurrency conflict exceptions. By combining the waiting lock and the invalid lock, CIODE can handle concurrency conflict exceptions and improve the execution efficiency of DNN applications.

Figure 2 shows the overall design of CIODE, which is divided into two phases, namely, the Planning phase and the Offloading phase. In the Planning phase, firstly, each edge node analyzes the structure of the DNN model and collects information, including estimated waiting time, network speed, available nodes, and prediction files. Secondly, according to the above information, CIODE builds a multiple edge nodes DNN collaborative execution graph, determines DNN layers that need to be uploaded, and records the corresponding target nodes and then sorts based on *delay improvement*. Finally, CIODE determines the offloading plan. In the Offloading phase, firstly, CIODE tries to upload and run the DNN model according to the offloading plan generated in the Planning phase. Secondly, CIODE refers to the real-time information returned by the waiting lock and the invalid lock to dynamically adjust the collaborative execution plan. We will introduce the Planning phase and Offloading phase with details in Sections 4.2 and 4.3.

4.2. The Planning Phase. The purpose of offloading DNN computing tasks is minimizing the execution overhead to improve the quality of service (QoS) of DNN applications. To this end, we design the CIODE Planning algorithm (Algorithm 1), which first divides the DNN model into partitions and then collects information, including estimated waiting time, network speed, available nodes, and prediction files. Due to the limited computing resource of edge nodes, it is difficult for a normal edge node to execute multiple DNN queries at the same time [34, 35]. We design the waiting lock in which an edge node can only execute one DNN query at a time; the other DNN queries are queued in order and return the estimated waiting time to the requester.

The estimated waiting time W_j of DNN query j is calculated by the following formula:

$$W_j = \sum_{i=0}^{j-1} (U_i + E_i), \tag{7}$$

where U_i is the layer upload time of query i and E_i is the predicted execution time of query i.

After considering the estimated waiting time, formula (5) can be updated as

$$D_{\text{offload}}(i) = T_{\text{input}}(i) + W_{\text{target}} + E_{\text{target}}(i) + U_{\text{target}}(i) + T_{\text{output}}(i), \tag{8}$$

where $D_{\text{offload}}(i)$ estimates the overall delay of offloading and running layer i. $T_{\text{input}}(i)$ and $T_{\text{output}}(i)$ are the data transmission times of input and output, respectively. W_{target} is the estimated waiting time on the target edge node. $E_{\text{target}}(i)$ is the predicted execution time of layer i on target edge node. $U_{\text{target}}(i)$ is the upload time of layer i on target edge node.

Secondly, the multiple edge nodes DNN collaborative execution graph is built. Next, the shortest path algorithm based on topological sorting is used to find the shortest path [36]. Then, DNN layers that need to be offloaded are determined and the target edge node corresponding to each layer is recorded. Here, the time complexity of this shortest path algorithm is $O(n)$ (n: the number of layers).

Then, the *delay improvement* of each layer is calculated. The *delay improvement* D_{impro} is calculated by the following formula:

$$D_{\text{impro}}(i) = D_{\text{local}}(i) - D_{\text{offload}}(i), \tag{9}$$

where $D_{\text{impro}}(i)$ estimates the delay improvement of offloading and running layer i. $D_{\text{local}}(i)$ is the local execution time of layer i. $D_{\text{offload}}(i)$ is the overall delay of offloading and running layer i.

Finally, the CIODE Planning algorithm takes *delay improvement* as the benchmark, gives priority to the DNN layer with the largest *delay improvement*, and generates the offloading plan.

The CIODE Planning algorithm is processed as Algorithm 1.

The input of Algorithm 1 is DNN models, prediction files, network speed, and trusted devices set. The output is offloading plan plan. Lines 1–11 represent the main function of the CIODE Planning algorithm. We first initialize the variables (line 1) and then collect the estimated waiting time returned by the waiting lock of other devices (line 2), the predicted execution time of each layer corresponding to different devices (line 3), and other information. In addition, we build a multiple edge nodes DNN collaborative execution graph based on the input information (line 4) and then find the shortest path and record the *id* of the DNN layer belonging to the requestee on the shortest path into *candidates* (line 5). Since the shortest path is calculated according to the weight of the path, the *candidates* have the *id* of the DNN layer that is beneficial for efficiency improvement, and the remaining layers will be executed locally. In addition, we sort the layers corresponding to each id in the candidates according to the *delay improvement* from largest to smallest (line 7). Because uploading the front layers of candidates first is most conducive to performance improvement, we use ppid to record the id of each target edge node (line 8). Then, we sequentially put the layer id and the corresponding ppid into the *plan* (line 9). Finally, return plan, the offloading plan (line 10).

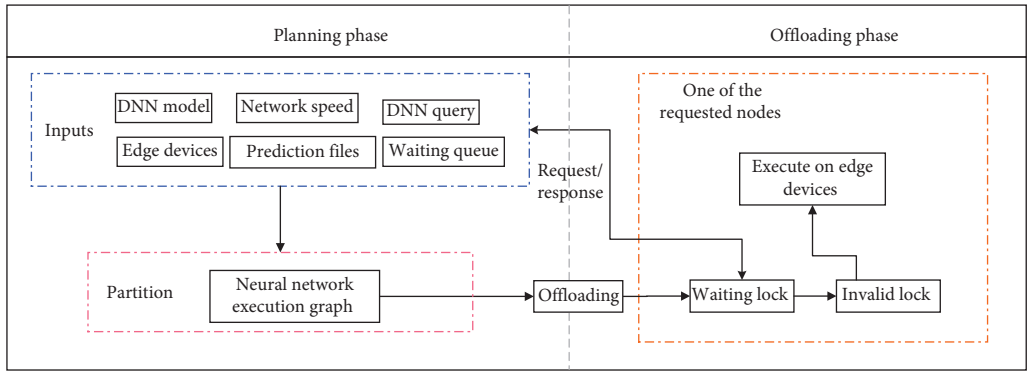

Figure 2: Overall architecture of CIODE.

(i) **Input** DNN models, prediction files, network speed, available devices set D (D_1, D_2,..., D_k);
(ii) **Output** plan;
0 **function** Planning:
(1) plan ⟵ [], candidates ⟵ [], ppid ⟵ [], wait ⟵ [], execute ⟵ [];
(2) update wait;//waiting queue of each device
(3) update execute;//the execution time of each layer corresponds to each device
(4) create the neural network execution graph ();
(5) candidates = shortest path ();
(6) ppid ⟵ device id;
(7) candidates = candidates.sort ();//Sort according to the *delay improvement*
(8) update ppid;
(9) plan = makepair (candidates, ppid);
(10) **return** plan;
(11) End;

Algorithm 1: The CIODE Planning algorithm.

4.3. The Offloading Phase. In this section, we upload and run the DNN model according to the offloading plan generated in the Planning phase. Then we design the CIODE Offloading algorithm (Algorithm 2), which includes the waiting period and the execution period.

During the waiting period, the CIODE Offloading algorithm will determine whether to replace the collaborative target according to formula (10). If formula (10) is true, our algorithm will replace the collaborative target. Formula (10) is as follows:

$$\min\{T_i(\text{others})\} < W_i(k) + U_i(k) + E_i(k), \tag{10}$$

where $W_i(k)$ is the estimated waiting time of DNN query i on device k. $U_i(k)$ is the layer upload time of i on device k. $E_i(k)$ is the predicted execution time of i on device k. $\min\{T_i(\text{others})\}$ estimates the minimum time of i to collaborate with new devices.

During the execution period, if the current uploading or executing query is not completed as expected but the next query starts to run as expected, a concurrency conflict exception will occur at this time. As the computing resources of edge nodes are very limited, concurrency conflict exceptions will cause a system deadlock, and the subsequent DNN requests will fall into a loop waiting. As a result, the inference efficiency of CIODE will be much lower than expected.

To address the above problem, we design an invalid lock for CIODE. The invalid lock will forcibly end the current query when formula (11) is true.

$$U_{\text{current}} + E_{\text{current}} > 2 \cdot (U_i + E_i), \tag{11}$$

where U_{current} is the current layer upload time. E_{current} is the current execution time. U_i is the predicted layer upload time of query i. E_i is the predicted execution time of query i.

That is, when a deadlock occurs, the occupying time would be much longer than predicted. Moreover, in a real executing environment, the time of uploading or executing might be longer than the predicted time. So, we set the upper limit of the occupied time to be twice the predicted execution time. If the time limit is exceeded, the invalid lock will forcibly end the relevant process. The predicted time of the uploading or executing query will be doubled and returned to the requester.

The CIODE Offloading algorithm is processed as Algorithm 2. Algorithm 2 inputs offloading plan plan, waiting queue wait, predicted execution time exe, and DNN request query and outputs the computation result result. First, during the waiting period, our CIODE Offloading algorithm determines whether to replace the collaboration device according to formula (10) (line 3). In addition, if formula (10) is true, replace the collaboration

```
(i) Input plan, wait, exe, query;
(ii) Output result;
0 function Offloading():
(1)   for i = 1 to plan.length
(2)     while wait[plan[i].second] > 0
(3)     if formula (10) = = true
(4)       update plan[i].second;//change collaboration device
(5)       offload(plan[i]);//upload partitions to the corresponding device
(6)       update wait;
(7)     end if
(8)   end while
(9) offload(plan[i]);
(10) end for
(11) result = execute(query);//cooperate with each corresponding device to compute
(12) return result;
(13) End;
```

Algorithm 2: The CIODE Offloading algorithm.

device, and upload DNN partitions to the new collaboration device and wait for execution (lines 4–6). Finally, execute the DNN query and return the result (lines 11–12).

An example scenario of the collaborative execution flow of a single layer is given on the left side of Figure 3. We describe the Offloading phase from the perspective of the smart glasses on the right side of Figure 3. First, after an edge node receives the request R, R enters the waiting lock. In addition, the waiting lock determines whether R is at the top of the waiting queue. If R is not at the top of the waiting queue, return the estimated waiting time to the smart glasses. At the same time, the smart glasses will determine whether to replace the collaboration device, and if there is a better choice, R will exit the current waiting queue and replace the collaboration device. If there is no better choice, R will continue to wait. If R is at the top of the waiting queue, R will enter the invalid lock and start offloading and execution. During offloading or execution, if a deadlock occurs, the execution delay will be updated and returned to the smart glasses, and the smart glasses will reconsider the offloading plan globally. Finally, if no deadlock occurs, offloading and execution continue.

5. Evaluation

In this section, we conduct experimental evaluations to validate the query efficiency, lock mechanism performance, and robustness of CIODE. We first compare the inference efficiency of different DNN execution strategies. Next, we evaluate the performance of CIODE's lock mechanism. Finally, we demonstrate the robustness of CIODE in heterogeneous network conditions.

5.1. Experimental Setup

5.1.1. Workloads. In experiments, we focus on comparing the inference efficiency of different DNN execution strategies in heterogeneous network conditions. In addition, we compare the performances of different schemes for handling concurrency conflicts and we assume that the client repeatedly proposes DNN queries for image classification. In this paper, each of posted results is the expectation value of 10 repeated experimental results.

5.1.2. Testbed. We implement CIODE on *Caffe* [37]. We use three edge devices to simulate multiple edge nodes DNN collaborative computing, namely, D1, D2, and D3. Among them, D1 is the request initiator, and D3 is a remote edge server that is deployed in another subnet. D2 and D3 assist D1 in executing the DNN query. They connect to our laboratory networks and communicate with each other using a network library *boost.asio*. Their detailed information is listed in Table 1. D1 has an i5-5200U CPU (2.20 GHz) and 8 GB memory. D2 has an i5-3470 CPU (3.20 GHz), NVIDIA GeForce GTX 1070, and 8 GB memory. D3 has an i7-9700 CPU (3.00 GHz), NVIDIA GeForce RTX 2060, and 16 GB memory. To evaluate the impact of heterogeneous network conditions, we set different network configurations, namely, Net-1 with a bandwidth of 80 Mbps and Net-2 with a bandwidth of 20 Mbps, and measure the current network speed before each offloading [38].

5.1.3. DNN Models. Alex-Net [39] was designed by Hinton, the winner of the 2012 ImageNet competition, and his student Alex Krizhevsky. Alex-Net uses an 8-layer neural network, 5 convolutional layers, and 3 fully connected layers and contains 630 million links, 60 million parameters, and 650,000 neurons. Res-Net [40] was proposed by Kaiming He et al. and won the championship in the ILSVRC2015 competition with the 3.57% error rate on top5. Res-Net can accelerate the training speed of neural networks, and the accuracy of the model is also greatly improved. VGG-16 [41] was the runner-up in the ImageNet ILSVRC challenge in 2014. It contains 16 conv/fc layers and features a homogeneous architecture that only performs 3×3 convolutions and 2×2 pooling from beginning to end.

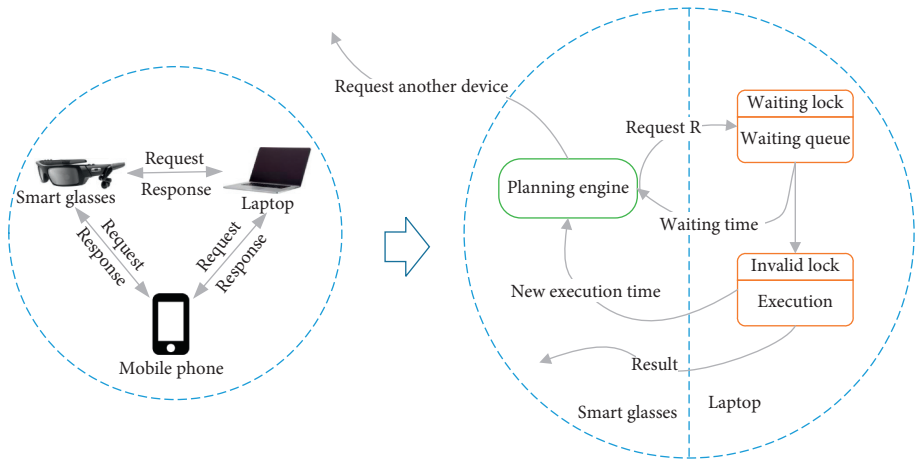

Figure 3: An example scenario of the collaborative execution flow of a single layer.

Table 1: Detailed information of edge devices.

Device	CPU	GPU	Memory (GB)
*D*1	i5-5200U (2.20 GHz 2.20 GHz)	No GPU	8
*D*2	i5-3470 (3.20 GHz 3.20 GHz)	GTX 1070	8
*D*3	i7-9700 (3.00 GHz 3.00 GHz)	RTX 2060	16

5.1.4. Implementation. We experiment with the above three DNN models. Then we use the four following DNN execution strategies to execute DNN queries: The first is All-at-once: upload the entire DNN model at once. The second is Enhanced IONN: it is the follow-up enhancement work of IONN [11], one-to-one computing offloading strategy based on an edge server. It partitions DNN models based on a penalty factor to reduce the uploading overhead; and it is the most popular approach evaluated in practice. The third is PerDNN [23]: it is a recent work on DNN offloading. It uses the GPU statistics of servers to partition DNN models to minimize the execution latency between the client and the edge server. The fourth is CIODE: it is our DNN collaborative computing strategy based on multiple edge nodes. We evaluate the efficiency by comparing the execution times of the above four DNN execution strategies (Section 5.2). Then, we evaluate the performance of CIODE's lock mechanism (Section 5.3). Finally, we experiment under jittery network conditions (Section 5.4) and compare their efficiencies under normal network conditions to reflect their robustness.

5.2. Efficiency Analysis. In order to evaluate the inference efficiency of CIODE, we compare CIODE with All-at-once, Enhanced IONN, and PerDNN in different DNN models with Net-1. A detailed result is listed in Table 2. With the exception of All-at-once, CIODE takes the least time to generate the offloading plan in different DNN models, which is less than 1/3 of Enhanced IONN and PerDNN. This is because, unlike the partitioning methods of Enhanced IONN and PerDNN, our CIODE Planning algorithm only calculates once to determine the offloading plan. In terms of time spent in offloading, CIODE achieves the best results in different DNN models. The time spent in the offloading of CIODE is shorter than those of the other three baselines. This is also because, instead of trying to upload more layers like other baselines, CIODE uploads layers on demand based on *delay improvement*. In classification, large models have more parameters that need to be transmitted between the edge server and the edge node, so their transmission time is much longer. CIODE avoids excessive transmission overhead by executing collaboratively on multiple edge nodes. So, the classification speed of CIODE on VGG-16 is faster. Similarly, since small models have fewer parameters to transmit, CIODE's speed advantage of transmitting parameters is insignificant. Therefore, CIODE achieves similar results to those of other baselines on Alex-Net and Res-Net. In general, CIODE achieves the best results in the time spent in the whole process.

Figure 4 shows the execution time of DNN queries in four cases: All-at-once, Enhanced IONN, PerDNN, and CIODE. The X value of each data point is the order when a DNN query is raised. The Y value is the time spent in executing the DNN query. All-at-once starts to offload the DNN execution only after the whole DNN is uploaded, so its query efficiency is low until the uploading is over. On the other hand, CIODE, Enhanced IONN, and PerDNN execute partial DNN tasks before the uploading is over, so the query efficiency is much better while uploading the DNN model. As shown in Table 2, CIODE requires the least time to generate the upload plan. Moreover, according to the offloading plan, the CIODE Offloading algorithm will upload the layer with the largest *delay improvement* first. We observe that the query execution time of CIODE rapidly decreases first and eventually reaches the minimal, which is the similar execution time of All-at-once, Enhanced IONN, and PerDNN when the uploading is over. It implies that CIODE can obtain efficiency improvement opportunities earlier than the other three baselines and can eventually achieve the highest efficiency.

Table 2: Time spent in each step with different models (seconds).

DNN models	Execution strategy	Planning	Offloading	Classifying	Total
Alex-Net	All-at-once	—	5.0584	0.1440	5.2024
	Enhanced IONN	$5.191E-05$	3.0320	0.1839	3.2160
	PerDNN	$4.107E-05$	2.8233	0.1961	3.0194
	CIODE	$1.302E-05$	2.2688	0.1882	2.4570
Res-Net	All-at-once	—	2.0166	0.0359	2.0525
	Enhanced IONN	$7.995E-05$	1.3463	0.0905	1.4369
	PerDNN	$7.024E-05$	1.4251	0.0976	1.5228
	CIODE	$2.211E-05$	0.6755	0.1041	0.7796
VGG-16	All-at-once	—	1.5674	0.1294	1.6968
	Enhanced IONN	$6.396E-05$	1.1425	0.1499	1.2925
	PerDNN	$6.508E-05$	1.0207	0.1502	1.1710
	CIODE	$1.591E-05$	0.7865	0.1417	0.9282

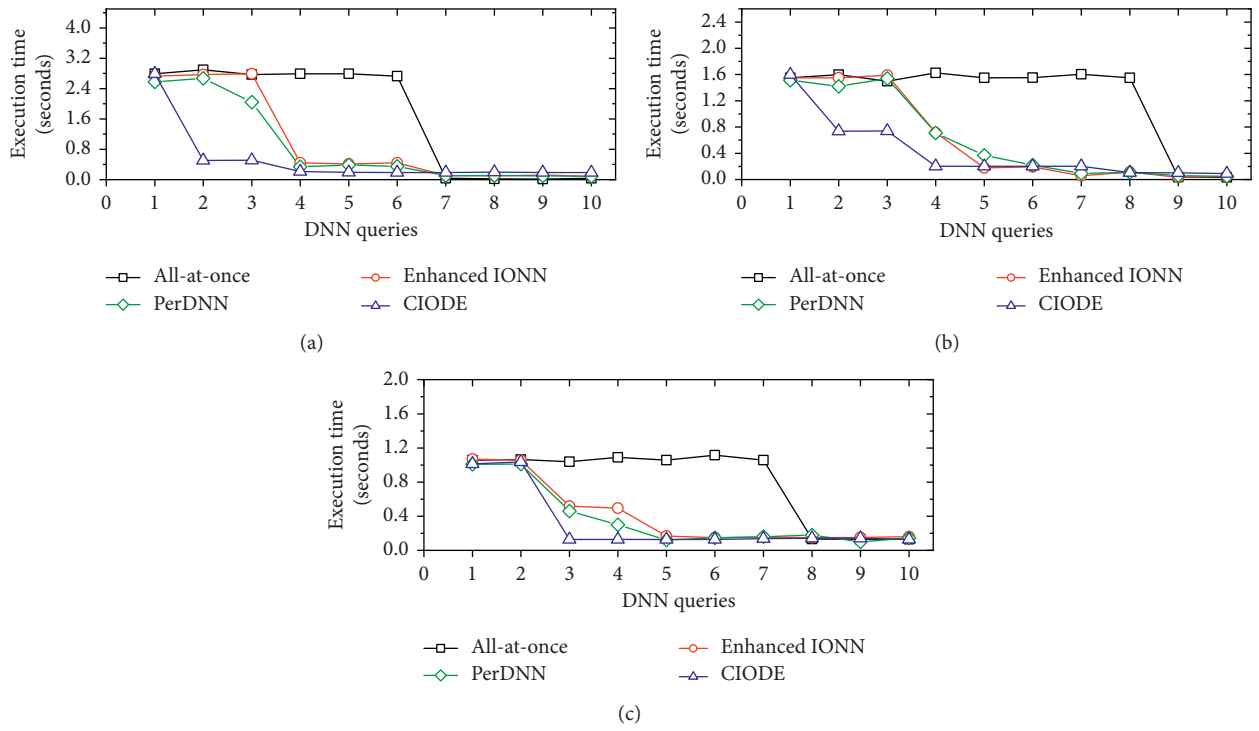

Figure 4: Execution time of DNN queries in different models. (a) Alex-Net. (b) Res-Net. (c) VGG-16.

5.3. Lock Mechanism Performance. Since other baselines have no mechanism to handle concurrent conflicts, in order to evaluate the performance of CIODE's lock mechanism, we compare CIODE with the Modified Binary Exponential Backoff (M-BEB) algorithm [42] in different DNN models with Net-1. M-BEB is a recently proposed scheme for determining backoff time after conflicts in multiple-access. In this scheme, the default backoff time is linearly increased or decreased after the request is successful or a conflict occurs, respectively.

We let D1 repeatedly propose DNN queries for collaborative execution when D2 was executing DNN queries locally. Figures 5(a) and 5(b) show the average execution time and the average conflicts of DNN queries in different models, respectively. Obviously, the average execution time of M-BEB is longer than that of CIODE (Figure 5(a)). This is because M-BEB has more conflicts than CIODE (Figure 5(b)). The reason why M-BEB has more conflicts is that M-BEB randomly waits for some time before reproposing the DNN query after a conflict occurs. However, conflicts may still occur when the DNN query is proposed again. On the contrary, our CIODE Offloading algorithm chooses to wait in queue after a conflict occurs instead of backing off and waiting for the next retry. Moreover, during the waiting period, the CIODE Offloading algorithm will also determine whether to replace the target device. In this way, CIODE can significantly reduce the average amount of conflicts (always less than 1). Similarly, CIODE's average execution time is also much less than that of M-BEB.

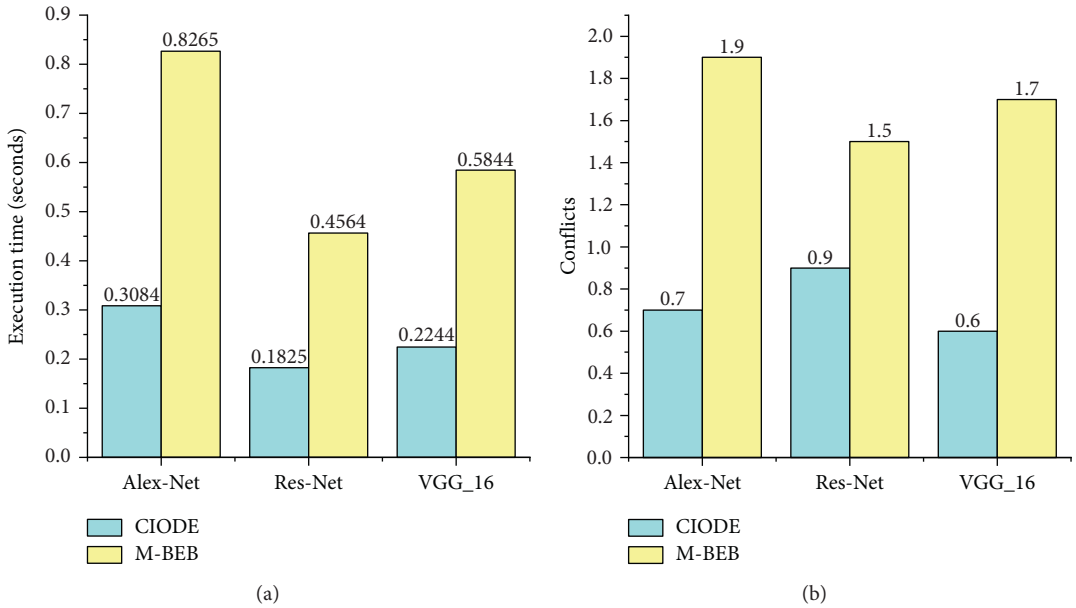

Figure 5: Comparison between Modified Binary Exponential Backoff (M-BEB) and CIODE. (a) Average execution time in different models. (b) Average conflicts in different models.

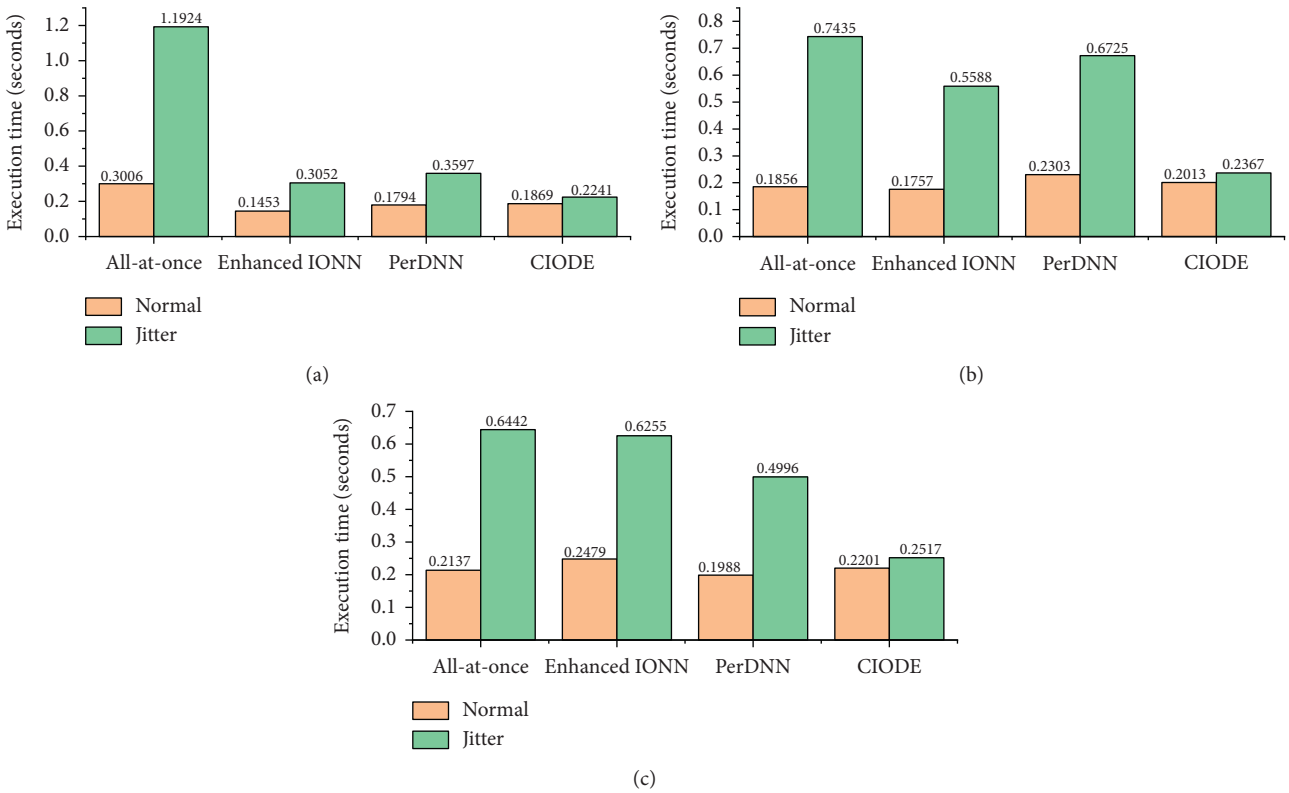

Figure 6: Comparison between normal and network jitters. (a) Alex-Net. (b) Res-Net. (c) VGG-16.

5.4. Robustness. In order to evaluate the robustness concern of each offloading scheme while a network jitter occurs, we take D3 to use Net-2, and others use Net-1. Then, we compare CIODE with All-at-once, Enhanced IONN, and PerDNN in different models.

Figure 6 shows how our CIODE adapts the change in heterogeneous network conditions. All-at-once, Enhanced IONN, and PerDNN are heavily impacted by changes in the bandwidth when the network between client (D1) and edge server (D3) is switched from Net-1 to Net-2. The execution

time of All-at-once increases by 296.7% in Alex-Net, by 300.6% in Res-Net, and by 201.5% in VGG-16. For Enhanced IONN, the execution time increases by 110% in Alex-Net, by 218% in Res-Net, and by 152.3% in VGG-16. PerDNN increases by 100.5% in Alex-Net, by 192% in Res-Net, and by 151.3% in VGG-16 when the network slows down. On the other hand, CIODE is not sensitive to network conditions. The average execution time increases by 17.3% when switching from the faster network to the slow one (Net-1 to Net-2). This is because the offloading plan generated by our CIODE Planning algorithm is based on multiple edge nodes rather than relying only on an edge server. The change in communication cost caused by network jitters between the client and the edge server has a lesser effect on the total execution time. It also indicates that CIODE takes better advantage of slower networks by cooperating with multiple edge nodes in order to improve efficiency.

6. Conclusions

In this paper, we propose the Conflict-resilient Incremental Offloading of Deep Neural Networks at Edge (CIODE) for improving the DNN inference efficiency at the edge smart environment. Since CIODE is based on multiple edge devices, CIODE can effectively avoid the influence of unstable factors and improve the stability of the system. We design a search algorithm based on graph theory to find the DNN layers that need to be uploaded, the upload order, and the collaborative devices on the set of trusted edge devices. We also design the waiting lock and the invalid lock to handle concurrency conflict exceptions and ensure the execution efficiency of DNN applications. Experimental evaluation shows that CIODE can improve the inference efficiency, handle concurrency conflict exceptions, and significantly improve the robustness of the system while ensuring performance. In short, CIODE realizes the possibility of running more computationally intensive applications in an edge smart environment by executing DNN applications in collaboration with multiple trusted edge nodes.

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 there are no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61962045, 61502255, and 61650205), the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2020-1-18), the National Key Research and Development Program of China (2018YFB1800403), the Strategic Priority Research Program of Chinese Academy of Sciences (XDC02030500), the Natural Science Foundation of Inner Mongolia Autonomous Region (2017MS(LH) 0601 and 2018MS06003), the Science and Technology Planning Project of Inner Mongolia Autonomous Region (2019GG372), the Key Technologies R&D Program of Inner Mongolia Autonomous Region (2020GG0094), the Science Research Project of Inner Mongolia University of Technology (BS201934), and the Visiting Scholar Project of China Scholarship Council (201908150030).

References

[1] A. Zielonka, A. Sikora, M. Woźniak, W. Wei, Q. Ke, and Z. Bai, "Intelligent-internet-of-things system for smart home optimal convection," *IEEE Transactions on Industrial Informatics*, vol. 16, 2020.

[2] J. Laufs, H. Borrion, and B. Bradford, "Security and the smart city: asystematic review," *Sustainable cities and society*, vol. 55, 2020.

[3] Y. Sako and M. Suzuki, "Automatic driving vehicle and program for automatic-driving vehicle," US Patent 10139824, 2018.

[4] D. R. J. G. J. Rydning, *The Digitization of the World from Edge Tocore*, Framingham: International Data Corporation, Framingham, MA, USA, 2018.

[5] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[6] N. Hassan, K.-L. A. Yau, and C. Wu, "Edge computing in 5g: a review," *IEEE Access*, vol. 7, pp. 127–276, 2019.

[7] Aliyun, http://www.aliyun.com, 2021.

[8] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 160–168, 2019.

[9] B. Charyyev, E. Arslan, and M. H. Gunes, "Latency comparison of cloud datacenters and edge servers," in *IEEE Global Communications Conference*, Madrid, Spain, December 2020.

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

[11] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM symposium on cloud computing*, pp. 401–411, Carlsbad, CA, USA, October 2018.

[12] K. Y. Shin, H.-J. Jeong, and S.-M. Moon, "Enhanced partitioning of dnn layers for uploading from mobile devices to edge servers," in *Proceedings of the 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, pp. 35–40, Seoul, South Korea, June 2019.

[13] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: resource-aware multitenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 115–127, New Delhi, India, May 2018.

[14] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Distributed perception by collaborative robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3709–3716, 2018.

[15] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits And Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[16] Z. Ning, K. Zhang, X. Wang et al., "Intelligent edge computing in internet of vehicles: a joint computation offloading and caching solution," in *IEEE Transactions on Intelligent Transportation Systems*, Rhodes, Greece, June 2020.

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

[18] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

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

[20] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, 2019.

[21] S. Gazzaz and F. Nawab, "Collaborative edge-cloud and edge-edge videoanalytics," in *Proceedings of the ACM Symposium on Cloud Computing*, p. 484, New York; NY, USA, March 2019.

[22] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 64–69, 2019.

[23] H.-J. Jeong, H.-J. Lee, K. Y. Shin, Y. H. Yoo, and S.-M. Moon, "Perdnn:offloading deep neural network computations to pervasive edge servers," in *Proceedings of the 2020 IEEE 40th International Conference On Distributed Computing Systems (ICDCS)*, pp. 1055–1066, Singapore, December 2020.

[24] Z. Xu, L. Zhao, W. Liang et al., "Energy-aware inference offloading for dnn-driven applications in mobile edge clouds," *IEEE Transactions on Parallel And Distributed Systems*, vol. 32, no. 4, pp. 799–814, 2020.

[25] R. Hadidi, J. Cao, M. S. Ryoo, and H. Kim, "Towards collaborative inferencing of deep neural networks on internet of things devices," *IEEE Internet of Things Journal*, vol. 25, 2020.

[26] Intel, "Movidius neural compute stick," 2021, https://software.intel.com/en-us/movidiusncs.

[27] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: local distributed mobile computing system for deep neural network," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 13–1401, Lausanne, Switzerland, May 2017.

[28] Nvidia, J., https://www.nvidia.com/en-us/autonomous-machines, 2021.

[29] Z. Xu, Z. Qin, F. Yu, C. Liu, and X. Chen, "Direct: resource-aware dynamic model reconfiguration for convolutional neural network in mobile systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 1–6, Seattle WA USA, July 2018.

[30] Q. Liang, P. Shenoy, and D. Irwin, "Ai on the edge: characterizing aibased iot applications using specialized edge architectures," in *Proceedings of the 2020 IEEE International Symposium OnWorkload Characterization (IISWC)*, pp. 145–156, Beijing, China, October 2020.

[31] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 854–863, Toronto, Canada, July 2020.

[32] 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.

[33] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques*, Addison-Wesley, Boston, MA, USA, 1986.

[34] Y. Jeon, H. Baek, and S. Pack, "Mobility-aware optimal task offloading in distributed edge computing," in *Proceedings of the 2021 International Conference on Information Networking (ICOIN)*, pp. 65–68, Jeju Island, Korea, January 2021.

[35] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 24, 2020.

[36] D. Epstein, "Ics 161: design and analysis of algorithms lecture notesfor," 1996.

[37] B. Vision and L. Center, Caffe, 2019.

[38] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Van Jacobson, "Behavioural Brain Research," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, Las Vegas, Nevada, USA, May 2016.

[41] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in Neuroscience*, vol. 13, p. 95, 2019.

[42] N. Zerguine, Z. Aliouat, M. Mostefai, and S. Harous, "Enhanced cand fair binary exponential backoff," in *Proceedings of the 2020 14th International Conference on Innovations in Information Technology (IIT)*, pp. 142–147, Saudi Arabia, UAE, November 2020.