DNN Migration in IoTs: Emerging Technologies, Current Challenges and Open Research Directions

Min Xue, Huaming Wu, *Senior Member, IEEE* and Ruidong Li, *Senior Member, IEEE*

***Abstract*—With the rapid development of the Internet of Things (IoT) and communication technology, Deep Neural Network (DNN) applications such as medical imaging, speech transcription, handwritten text recognition have been widely used in IoT devices. However, due to resource constraints on these devices, e.g., limited memory capacity, weak computing capacity, and low battery capacity, IoT devices cannot support complicated DNN operation effectively and thus fail to fulfill the requirements of Quality of Service (QoS) of mobile users. One promising approach is to migrate the DNN model to a remote cloud server to reduce the computing burden on IoT devices. Unfortunately, it still suffers from high delay and low bandwidth when communicating with cloud servers. Although the transmission delay of the edge server is low, its computing capacity lacks scalability and elasticity. To make matters worse, in the real world, the wireless connection between IoT devices and the cloud is intermittent, which can cause offloading failures during large-scale DNN data transmission. In this paper, we describe a DNN model migration framework to overcome the above challenges, which consists of three parts: DNN model preprocessing, partition-offloading plan, and partition-uploading plan. Accordingly, we introduce the operation of the DNN migration and the available methods for each part. In addition, we improve the DNN partition-uploading plan in a multi-user edge-cloud collaborative computing environment. Finally, we highlight the important challenges of achieving more efficient DNN migration and point out the unresolved issues of DNN migration, which may shed light on future research directions.**

***Index Terms*—DNN Partition, Computation Offloading, DNN Inference, Edge-Cloud Collaborative Computing, Intelligent Applications.**

I. Introduction

In recent years, the Internet of Things (IoT) and mobile Internet have developed rapidly, smart cities, smart homes, and smart transportation have become an indispensable part of social life. As the core of artificial intelligence, deep learning technology, especially Deep Neural Networks (DNN), has been widely used in a variety of fields, which also brings many new challenges to mobile terminals with limited resources. The requirements of emerging intelligent applications on communication systems are mainly reflected in four aspects: low response time, low energy consumption, low monetary cost, and balanced computing resource allocation. The most obvious contradiction is between the limited computing capacity of IoT devices and running complicated DNN inference. When DNN models implement inference in a resource-constrained computing environment, how to optimize objectives such as delay, energy, and monetary cost is of great significance.

Considering that the limited computing resources of mobile devices cannot support complex DNN operations, the traditional approach is to offload partial DNN models to remote cloud servers to reduce the pressure of local devices. DNN migration in edge, cloud or fog computing environment, as well as other computing platforms [1] has attracted great attention. As we all know, cloud computing has high computing capacity and sufficient storage space. However, the cloud server is usually far away from the local device, and the data transmission between the cloud server and the client is readily affected by various factors such as transmission delay, data volume, and central computing capacity. These factors put great pressure on the network bandwidth and easily cause excessive transmission delay, which cannot meet user Quality of Service (QoS) requirements. Compared with cloud servers, edge servers are closer to the client, which can effectively improve the efficiency of DNN offloading and reduce the pressure on network bandwidth. However, edge servers still suffer from limited computing capacity and insufficient storage.

Generally, it takes a long time to upload and execute the DNN model, due to the large size of the DNN model such as YOLO and FaceNet. Accordingly, DNN query, i.e., the process of DNN inference, will only be implemented on the client until the DNN model is uploaded. However, the client's computing capacity is low, which leads to poor performance of the DNN query. Thus, we perform the DNN query while uploading the DNN partition, and then use edge/cloud servers to optimize query performance. We usually use DNN migration to achieve two goals: one is to determine the DNN layer distribution on the edge/cloud server; the other is to determine the upload order of the DNN layer. Thus, how to obtain the optimal DNN migration scheme has become an urgent problem to be solved. In general, the migration of DNN models faces many challenges: 1) *Large-Scale DNN Partition:* In order to determine the upload order of the DNN layer, previous studies such as [2] have to divide the DNN model, but the DNN partition is usually very large, so there is no chance to realize more efficient DNN query performance through more fine-grained partitions. 2) *Poor Search Ability:* Most studies use traditional algorithms, e.g., swarm intelligence algorithms, to divide DNN models and obtain the DNN layer distribution under each server. However, these algorithms suffer from poor global search capabilities and tend to fall into local optimality. 3) *Simple Computing Environment:* DNN migration is only suitable for very simple environments, not for more realistic computing environments, such as multi-user multi-edge/cloud server environments. In addition, it is also difficult to achieve parallel processing of DNN models. 4) *User Requirements:* The requirements of users for QoS are constantly increasing, especially, obtaining the results of DNN migration with rel-

atively low delay and energy consumption at a lower cost. However, how to obtain an ideal cost on the premise of balancing time delay and the energy consumption is an urgent issue to be solved.

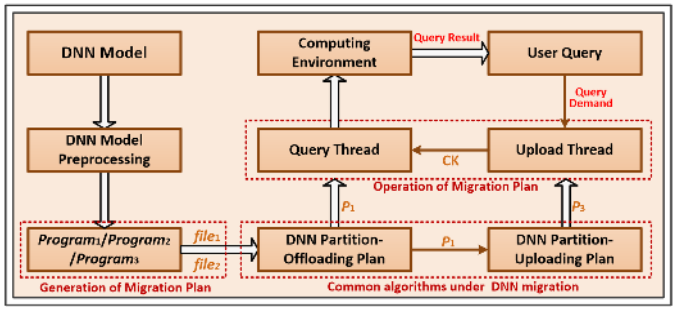

Fig. 1: The framework of DNN migration.

The framework of DNN migration is as shown in Fig. 1. Firstly, the edge-cloud collaborative computing environments can effectively integrate and utilize different types of computing resources, which is more suitable for large-scale DNN migration. Secondly, we expand the application of the DNN migration plan under the edge-cloud collaborative computing environment and realize the efficient partition-offloading plan and partition-uploading plan after DNN model preprocessing. Finally, we propose the generation and operation of DNN migration in a multi-server environment and introduced the information flow in it in detail.

II. Environments and Requirements for DNN Migration

A. Computing Environments of DNN Model

The IoT devices will generate a large number of DNN tasks, but they cannot handle large-scale DNN tasks, so some DNN tasks are often migrated to edge/cloud servers. We treat each DNN model as a task, and each layer in the DNN model as a subtask. Common computing environments usually include cloud computing environments, edge computing environments, and edge-cloud collaborative computing environments.

1) Cloud Computing Environment: According to the deployment form, the cloud can be divided into three types: private cloud, public cloud, and hybrid cloud. The traditional cloud computing environment has the advantages of strong computing capacity and sufficient storage space. However, because the cloud server is too far away from the IoT device, problems such as high transmission delay, network instability, and limited bandwidth are prone to occur, and thus it is difficult to meet the QoS requirements of users.

2) Edge Computing Environment: Any computing and network resource from the data source to the cloud computing center can be defined as the edge. Since the edge server is closer to the IoT device, it can respond to requests from the device in real-time, and the transmission is more secure. In addition, due to a large number of load flow nodes, the data transmission speed is fast, which is suitable for migrating DNN tasks that are highly sensitive to delay. However, the edge server has some defects such as small computing capacity and insufficient storage space.

3) Edge-Cloud Collaborative Computing Environment: Since IoT devices will generate a large number of DNN tasks, the edge-cloud collaborative computing environment, as a novel system architecture, can effectively accelerate the computational efficiency of DNN models. Fig. 2 shows the general diagram of the edge-cloud collaborative computing environment. Compared with the traditional cloud/edge computing environment, the edge-cloud collaborative computing environment owns the advantages of both the high transmission rate of edge computing and the high computing capacity of cloud computing, which can meet the migration requirements of the DNN model.

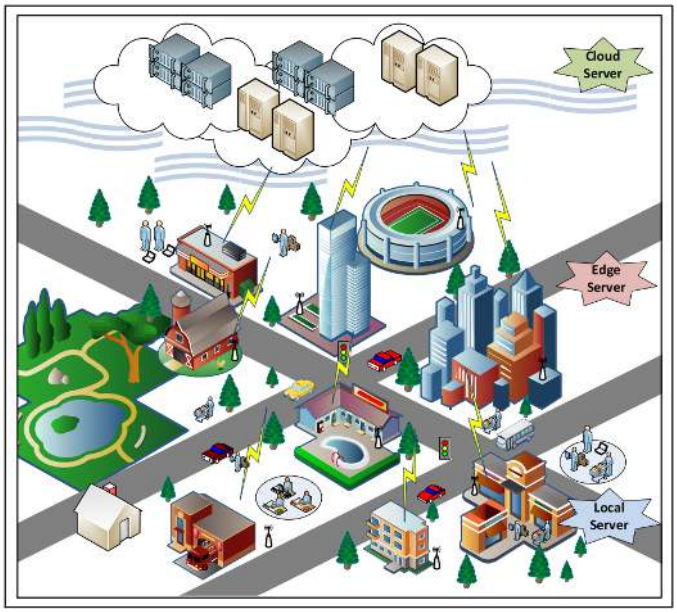

Fig. 2: The diagram of edge-cloud collaborative computing environments.

B. Migration Requirements of DNN Model

With the explosive development of IoT technology, the number of wireless devices has grown exponentially. In the meantime, a large number of delay-sensitive and computationally intensive applications are being deployed on these devices. The computing environment is required to provide powerful computing capabilities and transmission rates to ensure low-delay service quality. The migration of DNN models usually considers the effects of low delay, low energy consumption, and low monetary cost. In addition, the impact of network resource allocation is also taken into consideration.

III. Architecture of DNN Migration

The DNN migration plan includes DNN model preprocessing, DNN partition processing, DNN partition-offloading plan, and DNN partition-uploading plan, as shown in Fig. 3. The details are as follows:

DNN Model Preprocessing: It refers to compressing or merging DNN models before the DNN partition-offloading plan and partition-uploading the plan, thereby improving

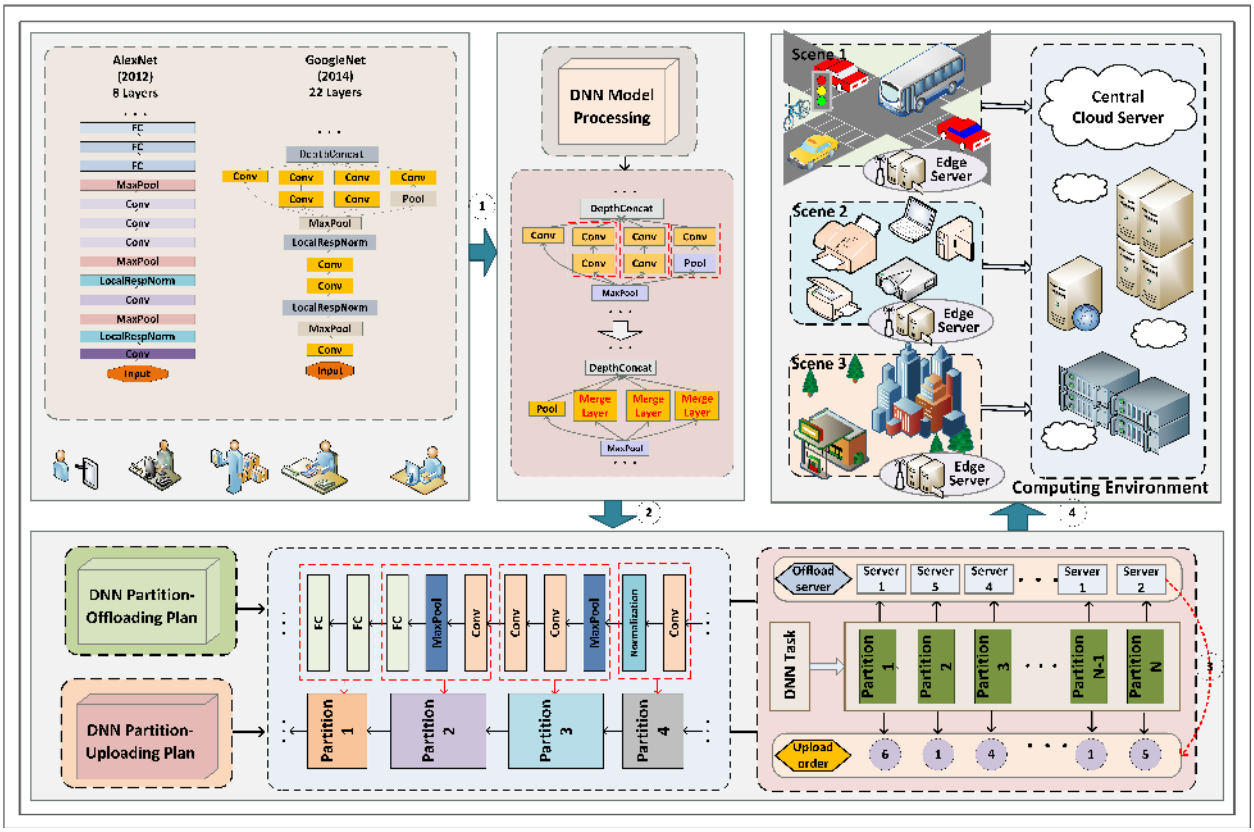

Fig. 3: The framework of DNN model migration. ① indicates the DNN model preprocessing, ② represents the partition-offloading and partition-uploading operations on the preprocessing results, ③ means that after the corresponding server for DNN subtask offloading is determined by the partition-offloading plan, the DNN partition-uploading plan obtains the upload order of DNN partition, ④ describes the uploading process of the DNN partition under the computing environment.

the processing efficiency of the DNN model. As shown in Fig. 3, it mainly includes inception module preprocessing, convolutional layer preprocessing, and fully connected layer preprocessing.

DNN Partition-Offloading Plan: As shown in Fig. 3, the DNN partitioning plan refers to splitting the DNN model, while DNN partition-offloading plan refers to specifying the corresponding offloading server for the DNN subtask. When the offloading location is determined by the partition-offloading algorithm, the partitioning result of the DNN model will be obtained naturally. We refer to the above algorithm as DNN partition-offloading algorithm.

DNN Partition-Uploading Plan: Due to the large size of the DNN model, if we directly upload the full DNN model to the cloud/edge server, this will lead to high transmission delay. Therefore, we consider performing DNN queries while uploading DNN partitions P_3. The DNN uploading plan refers to determining the sequence of uploading DNN partition to the edge/cloud server.

Existing DNN partition-uploading methods are mainly applied to the situation from a single local device to a single edge/cloud server. Most of the pioneering methods combine the shortest path, and after multiple iterations, the partitioning plan is generated while obtaining the partition-uploading plan.

It should be noted that in the DNN partition generated in the DNN partition-offloading plan, the DNN subtask deployed to the edge/cloud server is called P_1; the DNN subtask deployed on the client is called P_2. The DNN partition generated in the DNN partition-uploading plan is a further division of the DNN partition P_1 generated in the DNN partition-offloading plan. In other words, the DNN partition-uploading plan builds the upload sequence by dividing the DNN partition P_1 into a finer-grained DNN partition, namely P_3. Obviously, the DNN partition P_1 is composed of the DNN partition P_3.

A. *DNN Model Preprocessing:*

1) Inception Module Preprocessing: For the inception modules, it is common to consider splitting each inception module and executing them together on both local and edge/cloud servers. To avoid excessive data transmission on the wireless network, the inception module is partitioned only once, which can be considered as a minimum cut problem, where algorithms such as the Boykov-Kolmogorov maximum flow algorithm and the topological sorting algorithm can be applied. Then, the inception module is cut into two disjoint subgraphs, with the first one processed locally and the second one offloaded to the edge/cloud server [3]. In addition, when the difference between the predecessor's out-degree and the successor's in-degree under the inception module is 1, two adjacent layers are merged into a new layer. The data

dependency between the predecessor and the successor will disappear after preprocessing [1]. Some scholars have also proposed to directly treat the inception module as a whole [4].

2) Convolutional Layer Preprocessing: In view of the computationally intensive characteristics of the convolutional layer, the traditional convolution process is accelerated by low-rank decomposition schemes, and then the original kernel set in the convolutional layer is approximated by two low-rank decomposition kernels. In another way, the feature map of the convolutional layer is divided into blocks in space, and a related feature mapping block is assigned to an edge server, so that the feature map of the convolutional layer can be independently run on the edge server. In addition, the introduction of a 1×1 convolution kernel can effectively enhance the nonlinear expression ability of the network and greatly reduce the number of model parameters.

3) Fully Connected Layer Preprocessing: For the storage-intensive fully connected layer, the number of parameters in the fully connected layer is actually the size of the weight matrix. One method is to find a low-rank weight matrix to approximate the original weight matrix, thereby reducing the number of parameters of each fully connected layer, and then reducing the required storage space [5]. Another method is to use network pruning methods to sparse the network, reduce the over-fitting network, and then improve the generalization ability of the network [6].

B. DNN Partition-Offloading Plan:

1) One-Step Segmentation: It mainly refers to the processing of dividing the DNN model once and then offloading one of the partitions to the edge/cloud server. Some work considers the offload performance of partitioning at each candidate point, and selects the point with the optimal delay/energy consumption performance for partitioning [7]. Furthermore, some scholars have also proposed a binary search method, and then proved that the optimal partition-offloading decision follows a one-climb policy. Based on this, the Gibbs Sampling algorithm was proposed to obtain the optimal partition-offloading decision [8].

2) Swarm Intelligence Optimization: This method only relies on sampling the objective function, and then realizes the overall random search through a certain search mechanism, mainly including Simulated Annealing (SA), Particle Swarm Optimization (PSO) , and Genetic Algorithm (GA). In the edge-cloud collaborative computing environment, for each preprocessed DNN subtask, the optimal allocation of each subtask is searched as the partition-offloading strategy of the DNN model [1].

3) Distributed Deep Learning: Deep learning is a branch of machine learning that emphasizes learning from continuous layers. The distributed deep learning algorithm [9] closely combines distributed learning and deep learning, and uses multiple parallel DNNs to generate partition-offloading decisions, providing a new way for DNN partition-offloading decisions in an edge-cloud collaborative computing environment.

4) Deep Reinforcement Learning: Deep Reinforcement Learning (DRL) [10] reflects human learning by exploring and exploiting feedback from the environment. In order to achieve system optimization, previous works usually convert the DNN offloading process into a Markov Decision Process (MDP), and then select the appropriate DRL algorithm to achieve system optimization.

5) Meta Deep Reinforcement Learning: Since the sample complexity of meta-learning is closely related to DRL, it is considered to combine meta-learning and DRL in depth to obtain a meta-Deep Reinforcement Learning (meta-DRL) algorithm [11]. It can quickly adapt to new tasks and new computing environments, and then make partition-offloading decisions for the DNN model faster in an edge-cloud collaborative computing environment.

6) Deep Imitation Learning: Deep Imitation Learning (DIL) [12] is a traditional supervised learning method, which includes offline training and online decision-making. After the high-quality demonstration is generated, the model can be trained offline, and then the online decision can be made at a fast online inference speed, which can obtain the DNN partition-offloading plan very efficiently.

C. DNN Partition-Uploading Plan:

1) Direct Uploading Algorithm: This approach typically estimates the expected query execution time for each possible partition point and, based on that, finds the optimal partition point for the DNN model. By partitioning the DNN model once, we can obtain the DNN partition-uploading plan(i.e., partition-offloading plan), and then directly upload the corresponding DNN partition P_3(i.e., P_1) to the edge/cloud server [13].

2) IONN Algorithm: The DNN partition-uploading plan is obtained by using the shortest path method and penalty factor method. IONN uses up to eight penalty factors to find partitions P_3: $1, 0.5, 0.25, \cdots, 0.016, 0.0$. However, partitions P_3 are often very large, and there is a lack of opportunities for fine-grained partitions P_3 to provide better performance [14].

3) Efficiency-based Algorithm: The upload efficiency and the shortest path method are combined to find the next partition in each iteration, where the upload efficiency acts as a penalty factor. However, different from the fixed penalty factor, the upload efficiency has higher flexibility and selectivity. A greedy algorithm is designed based on the upload efficiency, and the partition P_3 with the highest upload efficiency is repeatedly selected to obtain a more fine-grained DNN partition-uploading plan [15].

4) Recursive Efficiency Algorithm: This approach recursively splits partitions obtained by the *efficient-based algorithm* and creates a more fine-grained DNN partition-uploading plan [15]. Compared with the *efficient-based algorithm*, it is obvious that this algorithm further improves the query performance of the DNN model.

IV. Generation and Operation of Migration Plan

A. Installation Phase: Generate Migration Plan

In this part, we will describe the installation of applications in an edge-cloud collaborative computing environment, briefly

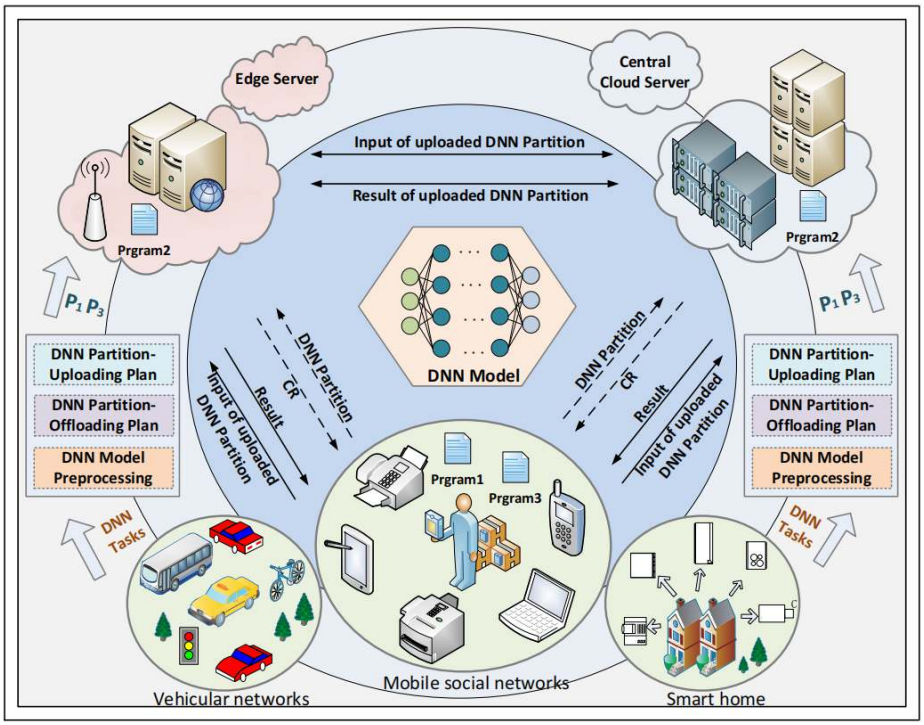

Fig. 4: The framework of information flow.

introduce the role of applications, and discuss the flow of information between applications.

From Fig. 4, the applications $Program_1$ and $Program_3$ installed on the client and the edge/cloud server, respectively, are used to obtain the execution time and index of DNN subtasks. The application $Program_2$ installed on the client is used to generate the migration plan. The information flow between the edge/cloud server and the client is as follows: First, we install $Program_1$ on the client, run each DNN subtask, and record the execution time and index of the DNN subtask as $file_1$. Then, the edge/cloud server cannot determine which DNN subtask will be executed, so the edge/cloud server cannot collect the execution delay of each DNN subtask like the client does. Install $Program_2$ on the edge/cloud server, where $Program_2$ uses regression functions to create prediction functions for DNN subtasks, which can estimate the execution delay of each DNN subtask under the edge/cloud server based on the parameters of the DNN and the server. Then, record the execution time and index of each DNN subtask as $file_2$. Finally, when the user enters the computing environment, $Program_1$ on the client runs and obtains $file_1$. At the same time, $Program_2$ will send the prediction results of each DNN subtask $file_2$ to the client. Then, $Program_3$ on the client combines $file_1$ and $file_2$ to create a migration plan.

B. Running Phase: Operate Migration Plan

Existing DNN partition-uploading methods are mainly applied to the situation from a single local device to a single edge/cloud server. We will execute the DNN query while uploading DNN partitions P_3, so two threads are required: (1) *Upload thread*: upload DNN model; (2) *Query thread*: execute real-time DNN query.

1) Upload Thread: The application $Program_2$ first creates a partition-offloading plan to obtain the DNN partition P_1 and its distribution on the edge/cloud server, and then creates a partition-uploading plan to obtain the DNN partition P_3 (P_3 is divided by the DNN partition P_1) and its upload sequence on the edge/cloud server. Then the *upload thread* starts to run the partition-uploading plan. First, the *upload thread* sends the first DNN partition P_3 in the partition-uploading plan from the client to the edge/cloud server, and sends the Confirmation Result (CR) of the DNN partition P_3 offloading back to the client. If the *upload thread* receives a failed result CR, it resends the DNN partition P_3 to the edge/cloud server. Conversely, if the *upload thread* receives a successful result CR, it sends the next DNN partition P_3 to the edge/cloud server, and then sends CR of the next DNN partition P_3 back to the client. Repeat the uploading process until the last DNN partition P_3 is uploaded to the edge/cloud server.

2) Query Thread: Consider performing DNN queries when uploading DNN partitions P_3. The client will repeatedly perform DNN queries, and immediately propose a new query after the previous query is completed. Before the DNN model is completely uploaded, the query can be executed jointly through the DNN partition P_2 under the client and the DNN partition P_3 under the edge/cloud server.

The *query thread* is the process of uploading DNN models, and performs a DNN query on the incomplete uploaded DNN model under the edge-cloud collaborative computing environment. The local device obtains which DNN partitions have been uploaded to the edge/cloud server by checking whether the CR of each DNN partition P_3 has arrived. When a DNN

query is triggered, the query thread will execute the DNN partitions P_3 in serial order. The first local DNN partition is performed, and then the input matrix (i.e., the output matrix of the first local DNN partition) and the index of the second DNN partition P_3 are sent to the designated client or edge/cloud server. After the second DNN partitions P_3 is executed, the input matrix and the index of the next DNN partitions P_3 will be sent to the client or the edge/cloud server where the next DNN partitions P_3 is located. In this way, the client and the edge/cloud server jointly execute the DNN partition P_3 until the whole DNN model is executed.

V. Improvement and Comparison

A. Improvement of Partition-Uploading Plan

It is known that the existing partition-uploading algorithm is only suitable for the situation from a single client to a single edge/cloud server, which is obviously not suitable for DNN uploads in real scenarios. In this part, we will discuss the performance of the DNN partition-uploading plan in a multi-user edge-cloud collaborative computing environment. It is mainly divided into the following two steps:

- Firstly, we simply apply the PSO algorithm to obtain the corresponding offloading position of the DNN subtasks in a multi-user edge-cloud collaborative environment.
- Secondly, according to the existing partition-upload algorithm, combined with the unloading position of the DNN subtask, the DNN partition P_1 is divided to obtain the DNN partition P_3, and the upload order of each DNN partition P_3 can be determined.

B. Experimental Comparison

1) Experimental Setup: We build an edge-cloud collaborative computing environment $\mathbb{R} = \{r_1, r_2, \cdots, r_{12}\}$, where the first two belong to the clients, the last five belong to the cloud servers, and the remaining five belong to the edge servers. We set the bandwidth between the local and the edge is 10 MB/s, between the local and the cloud is 0.5 MB/s, between the edge and the cloud is 0.5 MB/s, between different cloud servers is 5 MB/s and between different edge servers is 10 MB/s. The CPU processing capacity of the client, the edge server, and the cloud server are set to $1.1 \sim 2.3$ GHz, $4.2 \sim 18.3$ GHz, and $40 \sim 120$ GHz, respectively. Moreover, the model size of Alexnet is 223 MB, and the model size of the VGG model is 548 MB.

2) Performance Comparison and Analysis: We apply the classic *Direct Uploading algorithm*, *IONN algorithm* and *Recursive Efficiency algorithm* to the multi-user edge-cloud collaborative computing environment, so as to illustrate the feasibility and necessity of the DNN partition-uploading plan in the multi-server environment.

As shown in Fig. 5, we track the system delay changes of repeated queries of the Alexnet model and VGG model, respectively. Among them, the horizontal axis represents the upload delay of the DNN partition P_3, and the vertical axis represents the change of the system delay with the upload of the DNN partition P_3. We consider performing DNN queries while uploading DNN partitions. In this process, we assume that the client will repeatedly put forward DNN queries, and execute new queries immediately after the previous one is completed, until the entire model is uploaded. In addition, During the upload period of the DNN partition P_3, the DNN system delay will not change during the upload period of the DNN partition P_3.

When compared with the *Direct Uploading algorithm* that does not adopt the DNN partition-uploading plan, the *IONN algorithm* and *Recursive Efficiency algorithm* using the DNN partition-uploading plan have a lower total system delay to complete the DNN model query. This is mainly due to the DNN partition P_1 being too large and the transmission delay of uploading the whole DNN partition P_1 being too high. In the process of uploading the DNN partition P_1, DNN queries will only be completed by local devices, and the insufficient computing capacity of local devices leads to the high delay of DNN queries. In addition, the *Recursive Efficiency algorithm* has better query results than that of *IONN algorithm*, which is mainly obtained by more fine-grained partitions P_3. In summary, we know that adopting the DNN partition-uploading plan can obtain more efficient DNN query performance, and more fine-grained DNN partition P_3 helps to obtain more efficient DNN query results. In short, the above experimental results demonstrate that the method proposed in this section can extend different types of DNN partition-uploading plans to a multi-server environment. Furthermore, the comparison with the *Direct Uploading algorithm* also illustrates the necessity of extending the DNN partition-uploading plan to a multi-server environment.

VI. Current Challenges and Future Directions of DNN migration

Along with the increasing number of IoT devices, people have new demands for service quality in daily life. In addition to the large granularity of DNN partitions and not being suitable for multi-server environments, DNN migration also faces some challenges and opportunities in real-world scenarios.

On the one hand, applications such as Unmanned driving, Virtual Reality (AR) and Augmented Reality (AR) all put forward strict requirements for the delay. However, processing massive amounts of data on the IoT devices puts a heavy burden on battery consumption, and the high data transmission delay of the cloud computing center also brings greater transmission energy consumption pressure on the client. Furthermore, The requirements of users for QoS are constantly increasing, and they hope to obtain the results of DNN migration at a lower cost. In summary, when DNN migration is performed in a resource-limited environment, how to achieve the overall system optimization of latency, energy consumption, and cost and how to specifically adjust the optimization degree of each optimization objective according to user requirements is an urgent issue to be solved.

On the other hand, we consider practical issues that are ignored in the process of DNN model migration. These issues are caused by the greedy occupation of resources by users and the unbalanced metric optimization. It is known that the users

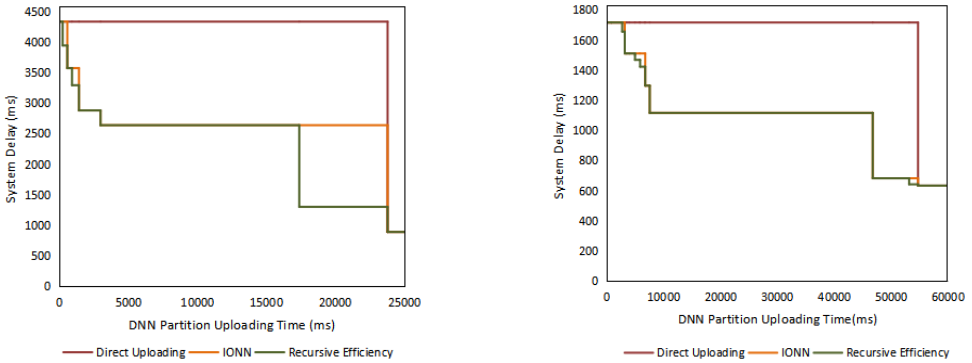

Fig. 5: Comparison of various DNN partition-uploading methods. (a) Alexnet model. (b) VGG model.

entering the computing environment are constantly changing, so when the users entering the environment first occupy all the computing resources of some edge/cloud servers, the users entering the environment later will not be able to use the above servers, resulting in an unreasonable allocation of computing resources. It can be seen that the unreasonable allocation of computing resources caused by the dynamic changes of the computing environment is also a problem worth studying in the DNN migration in reality.

Most importantly, it is necessary to consider a more practical circumstance of migrating large-scale DNN subtasks in a more complex and resource-limited computing environment with multiple users in future work. Among others, researchers should consider whether the proposed DNN partition-offloading plan and DNN partition-offloading plan are suitable for DNN models with different types, sizes, and structures, which is an issue that researchers need to focus on.

VII. Conclusion

In this paper, we have given our vision and explored numerous new trends for DNN migration in IoTs. To fulfill the requirements of intelligent applications for communication systems in a computing environment with limited resources, we have designed a DNN model migration framework, where the DNN migration plan consists of three parts, including DNN model preprocessing, partition-offloading plan, and partition-uploading plan. We first describe the computing environment and migration requirements for the DNN model. Then, we comb the generation and operation of DNN migration and summarize the common algorithms of DNN migration. Furthermore, we propose a DNN partition-uploading plan in a multi-user edge-cloud collaborative computing environment and compare the performance of different partition-uploading plans. Finally, we consider the prospects and challenges of DNN migration under the existing technical conditions.

Edge computing provides low-latency, high-availability and privacy protection for local computing services, and solves the problems of high latency of cloud computing and constraints by network environment. Accordingly, the DNN migration or placement technology will vigorously mitigate the contradiction between the limited computing capacity of IoT devices and complex DNN inference. We expect that this visionary research will be helpful to practitioners and researchers who are interested in doing research on the DNN migration technology to promote intelligent applications, smart cities, and other industrial upgrades.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 62071327 and JSPS KAKENHI under Grant No. 19H04105.

References

[1] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge, and end devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5456–5466, 2020.

[2] 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.

[3] X. Tian, J. Zhu, T. Xu, and Y. Li, "Mobility-included dnn partition offloading from mobile devices to edge clouds," *Sensors*, vol. 21, no. 1, p. 229, 2021.

[4] L. Lockhart, P. Harvey, P. Imai, P. Willis, and B. Varghese, "Scission: Performance-driven and context-aware cloud-edge distribution of deep neural networks," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, dec 2020.

[5] A. Ss, B. Dg, C. Rk, and D. Fa, "Sparse low rank factorization for deep neural network compression," *Neurocomputing*, vol. 398, pp. 185–196, 2020.

[6] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Mar. 2020.

[7] C. Ding, A. Zhou, Y. Liu, R. Chang, and S. Wang, "A cloud-edge collaboration framework for cognitive service," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2020.

[8] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 235–250, 2020.

[9] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Networks and Applications*, 2018.

[10] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 102, pp. 847–861, 2020.

[11] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[12] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 92–99, 2020.
[13] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGPLAN Not.*, vol. 52, no. 4, p. 615–629, Apr. 2017.
[14] 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*. ACM, Oct. 2018.
[15] K. Y. Shin, H.-J. Jeong, and S.-M. Moon, "Enhanced partitioning of DNN layers for uploading from mobile devices to edge servers," in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications - EMDL '19*. ACM Press, 2019.

Min Xue is currently working towards a Master's degree at the Center for Applied Mathematics, Tianjin University, China. Contact her at xm_17@edu.cn.

Huaming Wu is currently an associate professor at the Center for Applied Mathematics, Tianjin University, China. Contact him at whming@tju.edu.cn.

Ruidong Li is currently an Associate Professor in Institute of Science and Engineering, Kanazawa University, Japan. Contact him at liruidong@ieee.org.