DDPQN: An Efficient DNN Offloading Strategy in Local-Edge-Cloud Collaborative Environments

Min Xue, Huaming Wu, *Member, IEEE*, Guang Peng, and Katinka Wolter

Abstract—With the rapid development of the Internet of Things (IoT) and communication technology, Deep Neural Network (DNN) applications like computer vision, can now be widely used in IoT devices. However, due to the insufficient memory, low computing capacity, and low battery capacity of IoT devices, it is difficult to support the high-efficiency DNN inference and meet users' requirements for Quality of Service (QoS). Worse still, offloading failures may occur during the massive DNN data transmission due to the intermittent wireless connectivity between IoT devices and the cloud. In order to fill this gap, we consider the partitioning and offloading of the DNN model, and design a novel optimization method for parallel offloading of large-scale DNN models in a local-edge-cloud collaborative environment with limited resources. Combined with the coupling coordination degree and node balance degree, an improved Double Dueling Prioritized deep Q-Network (DDPQN) algorithm is proposed to obtain the DNN offloading strategy. Compared with existing algorithms, the DDPQN algorithm can obtain an efficient DNN offloading strategy with low delay, low energy consumption, and low cost under the premise of ensuring "delay-energy-cost" coordination and reasonable allocation of computing resources in a local-edge-cloud collaborative environment.

Index Terms—Mobile edge computing, cloud computing, QoS, computation offloading, DNN partition

1 Introduction

DEEP learning technologies, especially Deep Neural Networks (DNN), break through the bottleneck of traditional machine learning and have been widely applied in mobile applications and services, ranging from natural language processing, autonomous driving, biomedicine to image processing. More and more Artificial Intelligence (AI) applications, e.g., face recognition, Virtual Reality (VR), and Augmented Reality (AR), are being deployed on mobile and Internet of Things (IoT) devices, which poses many new challenges to mobile systems. The most obvious one is the contradiction between the limited computing capacity of IoT devices and running complex DNN inference, which cannot be solved in a short time due to the slow hardware development in small-sized equipment.

Computation offloading determines which subtasks should be offloaded to cloud servers or edge servers for execution, and which subtasks need to be processed locally [1]. It can effectively solve the deficiencies in computing capacity, storage, and energy consumption for clients. Deep Reinforcement Learning (DRL) can achieve efficient offloading performance in computation offloading [2]. The DRL algorithm combines the perception ability of deep learning and the decision-making ability of reinforcement learning, using the agents to continuously interact with the environment to learn the optimal actions to take in different states to maximize rewards. This high-dimensional search algorithm for maximizing rewards is suitable for computation offloading [3].

Facing the challenge that the constrained computing resources of mobile devices are insufficient to support complex DNN inferences, the traditional approach is to offload part of the DNN model to cloud servers, so as to alleviate the computing pressure of clients. Cloud computing [4] can provide users with abundant computing resources and storage resources. However, cloud servers are generally far away from mobile devices, and the massive data transmission between the device and cloud is easily affected by factors, e.g., network bandwidth, data volume, central computing capacity, and transmission delay [5]. This brings a huge squeezing force to the network bandwidth, which can easily cause excessive delay and fails to meet the user's requirements for Quality of Service (QoS). Along with the rapid development of edge computing technology, DNN applications can also be deployed on edge servers. Compared with cloud servers, edge servers are closer to the data source, which can greatly reduce the pressure on network bandwidth while improving the efficiency of DNN processing [6]. Nevertheless, edge servers still have shortcomings, e.g., limited computing power and insufficient memory [7].

In recent years, the research on DNN partitioning and offloading in the cloud/edge computing environments has attracted more and more attention, providing several solutions in different directions. First, the client still suffers from high transmission delay when communicating with distant cloud servers, while edge servers have lower transmission delay, but their computing capability is not scalable. Most studies [8], [9], [10] have not considered how to implement

- *Min Xue and Huaming Wu are with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China. E-mail: {xm_17, whming}@tju.edu.cn.*
- *Guang Peng and Katinka Wolter are with the Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany. E-mail: {guang.peng, katinka.wolter}@fu-berlin.de.*

Manuscript received 30 Mar. 2021; revised 10 Aug. 2021; accepted 27 Sept. 2021. Date of publication 30 Sept. 2021; date of current version 8 Apr. 2022.
This work was supported by the National Natural Science Foundation of China under Grants 62071327 and 61801325 and the Research and Innovation Project for Postgraduates in Tianjin (Artificial Intelligence) under Grant 2020YJSZXS27.
(Corresponding author: Huaming Wu.)
Digital Object Identifier no. 10.1109/TSC.2021.3116597

DNN offloading in a local-edge-cloud collaborative environment where the computing resources of IoT devices are limited. Moreover, most research conducted on DNN offloading is focusing on reducing delay and saving energy consumption of IoT devices, while neglecting to minimize the cost. Lastly but most importantly, researchers have ignored the problems caused by the incoordination between delay, energy consumption, and cost, as well as the unreasonable allocation of resources. The details are as follows: i) In some cases, the system evaluation performance is very good, but one of the evaluation indicators, e.g., delay, energy consumption, and cost, is too high or too low. ii) It often happens that the user enters the environment first occupies all the computing resources of some servers, and the user enters the environment later cannot use the above-mentioned servers, resulting in unreasonable resource allocation.

To address the above challenges, we consider achieving the goals of low delay, low energy consumption, and low cost in a local-edge-cloud collaborative environment with limited resources. A novel Double Dueling Prioritized deep Q-Network (DDPQN) algorithm is proposed to obtain the optimal DNN offloading strategy. In addition, we consider the queuing time caused by the offloading of large-scale DNN models in a resource-constrained environment. Then we introduce the coupling coordination indicator and node balance indicator to optimize the coordination of delay, energy, and cost, and the rationality of resource allocation, so as to achieve an efficient DNN offloading strategy and meet QoS requirements. To the best of our knowledge, *DDPQN* is the first work to formally model the rational allocation problem of computing resources and the coordination optimization problem between delay, energy, and cost, which realizes the efficient partitioning and offloading of large-scale DNN models based on the DRL algorithm. The main contributions of this paper can be summarized as follows:

- A comprehensive DNN partitioning and offloading strategy for IoT systems is designed, which takes the coordination between delay, energy consumption, and cost into account. The setting of the coupling coordination degree can effectively reduce the phenomenon that the system of multiple indicators is excellent but the individual indicators differ greatly.
- Considering the unreasonable distribution of computing resources caused by some users greedily occupying the server resources, we effectively improve the utilization of computing resources by optimizing the balance of the DNN layer distribution in the local-edge-cloud collaborative environment, i.e., node balance.
- Considering the parallel offloading of large-scale DNN models, we propose a *DDPQN* algorithm in a local-edge-cloud collaborative environment under the constraints of parallel pools, which obtains an efficient DNN offloading strategy with low delay, low cost, and low energy consumption.

2 Related Work

In recent years, more and more attention has been paid to the research of efficient DNN offloading, aimed at finding the optimal offloading strategy with low delay or energy consumption. Scholars have conducted in-depth research on DNN partitioning and offloading, and proposed feasibility studies from various aspects.

2.1 Edge/Fog/Cloud-Based DNN Offloading

Jeong *et al.* [11] applied the shortest path and penalty factor to divide and offload the DNN model from a single client to a single edge server. Li *et al.* [12] joined DNN partition and DNN right-sizing to maximize precision, while ensuring application delay requirements. Qi *et al.* [13] proposed a model scheduling algorithm that adaptively selects the cloud or mobile terminal according to the terminal status and network status. Yu *et al.* [14] applied deep imitation learning based on mobile edge computing (MEC) to minimize offloading costs. Hu *et al.* [15] designed a Dynamic Adaptive DNN Surgery (DADS) scheme to segment the DNN model and accelerate DNN inference. Kang *et al.* [16] considered how to effectively leverage the cloud and client cycle acquisition DNN partitioning strategy in the cloud computing environment to achieve low delay, low energy consumption, and high data throughput for the application. Considering DNN partitioning optimization, Wang *et al.* [17] proposed an adaptive distributed scheme to accelerate DNN inference, which can realize dynamic offloading of DNN according to the edge computing environment. Tang *et al.* [18] introduced an Iterative Alternating Optimization (IAO) algorithm, which considers the relationship between DNN partitioning and resource allocation under limited resource conditions. Ju *et al.* [19] proposed a DeepSave scheme in the hope of saving more frames for DNN inference during switching. Mohammed *et al.* [20] considered a distributed offloading scheme (DINA) based on the matching game method for DNN partition and offloading in a fog environment.

2.2 Local-Edge-Cloud Collaborative DNN Offloading

Teerapittayanon *et al.* [21] proposed a distributed DNN over the client, edge server, and cloud server. This method can not only perform DNN inference on the cloud, but also use the shallow layer of neural networks to perform fast local inference on the edge and clients. Ren *et al.* [22] considered a collaborative object recognition solution for mobile Web AR based on the edge, and explored more fine-grained DNN partitions under the mobile web browser, cloud, and edge. Chen *et al.* [23] proposed a method based on the greedy and genetic algorithm to optimize the average response time of multi-task parallel scheduling. Ding *et al.* [24] used CloudCNN to assist in training EdgeCNN so that it could provide persistent and rapid response cognitive services. Pachecom *et al.* [25] accelerated DNN inference by dividing DNN between edge server and cloud server to support high-response programs. In addition, combining edge and cloud computing resources, network bandwidth, and inherent data parameters, a DNN partitioning optimization solution is proposed. Considering that the mobility of mobile devices is prone to computing offload failure, Tian *et al.* [26] proposed to offload part of the DNN model to the edge/cloud to realize the cooperative execution between the mobile device, edge server, and cloud server. Based on

this, a DNN partition offloading algorithm (MDPO) for mobile users is developed, which can reduce the total delay of the DNN model to the greatest extent when users are moving.

2.3 A Qualitative Comparison

We briefly analyze the problems that existed in the previous work in the above environment. Then, in view of the shortcomings of the existing work, improvement measures are proposed.

To optimize the delay or energy consumption of DNN inference, most of the previous work is to offload the DNN model in an edge/fog/cloud computing environment or a local-edge-cloud computing environment with a single edge server and a single cloud server. First, In the above computing environment, computing resources are usually limited, which is not suitable for offloading large-scale DNN models. In addition, researchers also ignore the queuing time caused by limited computing resources. Second, the methods in the previous work are usually suitable for successively implementing DNN offloading, which cannot achieve parallel offloading of DNN models and is inappropriate for large-scale DNN parallel offloading. Third, when multiple factors of delay, energy consumption, and cost are used as optimization targets, the coordination among multiple factors is usually ignored. In other words, the system of multiple factors is optimized, but one of the factors is optimized too high, and one of the factors is optimized too low. Finally, we find that the previous work rarely considered the unreasonable allocation of computing resources caused by users first enter the computing environment occupies all resources of some servers.

In this paper, we propose to achieve the goals of low delay, low energy consumption, and low cost. First, compared with existing work, we consider a local-edge-cloud collaborative environment with multiple edge servers and cloud servers, which is suitable for large-scale DNN model offloading. We also consider the queuing time caused by the limited resources of the computing environment when offloading the large-scale DNN model. Second, combined with the DRL algorithm, we develop the *DDPQN* algorithm, which is a high-dimensional search algorithm for maximizing rewards and is suitable for parallel offloading of large-scale DNN models. Third, to avoid the problem of excessive differences in the optimization of various indicators, we introduce the coupling coordination indicator to realize the coordinated optimization of delay, energy consumption, and cost. Finally, considering the unreasonable allocation of computing resources, we propose to use the node balance indicator to optimize the above problem.

3 System Model and Problem Definition

In this section, we first propose *two-step offloading strategy* to meet the offloading requirements of the chain type DNN model and the topology type DNN model containing the Inception Module. Then we introduce the local-edge-cloud collaborative environment and formulate the DNN offloading model. The generation of the DNN offloading strategy is shown in Fig. 1. In addition, taking into account the dimensions of different indicators, we standardize the data before the start of the offloading plan.

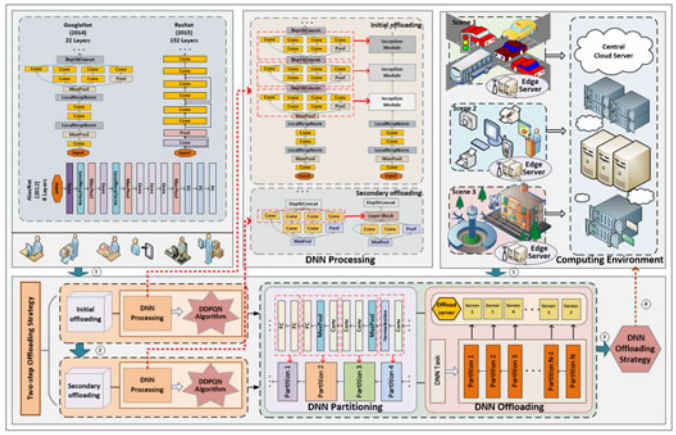


Fig. 1. A DNN offloading strategy in a local-edge-cloud collaborative environment. ① indicates input DNN task parameters and local-edge-cloud collaborative environment parameters, ② represents that the secondary offloading will start after the initial offloading is completed, ③ indicates the generation of DNN offloading strategy, and ④ denotes offloading the DNN partitions to the local-edge-cloud collaborative environment.

3.1 DNN Model Preprocessing

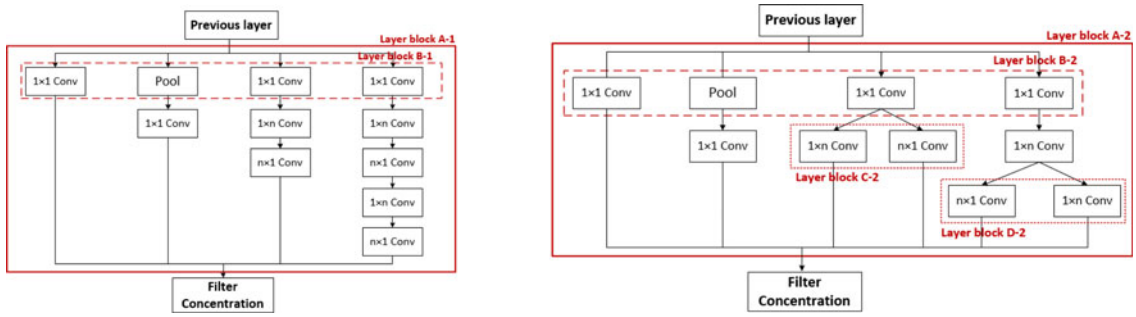
For the offloading of the Inception Module, our principle is to treat the layer with the same input as a whole. We elaborate on two types of Inception modules in Fig. 2. The *two-step offloading strategy* for the Inception Module is conducted as follows:

- *Initial offloading*: we regard the Inception Module as one layer in the DNN model, and perform a preliminary partitioning and offloading of the DNN model. After *initial offloading*, we can get the offloading position of the input layer and output layer of the Inception Module.
- *Secondary offloading*: Based on *initial offloading*, we divide the Inception Module and offload each branch of the Inception Module separately. At this time, we note that the final offloading delay of the Inception Module is the branch that takes the longest time, and the energy consumption and cost are the sums of all branches, respectively.

3.2 System Overview

We design a local-edge-cloud collaborative environment with multiple edge servers and multiple cloud servers. The client has problems such as insufficient resources and poor device performance. Therefore, we combine the advantages of cloud servers with scalable computing capabilities and edge servers with low transmission delay to support efficient offloading of complex DNN applications [27].

The local-edge-cloud collaborative environment $\langle \mathbb{L}, \mathbb{E}, \mathbb{C} \rangle$ is composed of client, edge and cloud, where $\mathbb{L} = \{L_1\}$ consists of one client, $\mathbb{E} = \{E_1, E_2, \ldots, E_n, \ldots, E_b\}$ consists of b edge servers, and $\mathbb{C} = \{C_1, C_2, \ldots, C_m, \ldots, C_d\}$ consists of d cloud servers. The environment can also be described as $\mathbb{M} = \{M_1, M_2, \ldots, M_s, \ldots, M_{(1+b+d)}\}, s \in [1, 1+b+d]$. Among them, M_1 is the client $\mathbb{L}$, $\{M_2, M_3, \ldots, M_{(1+b)}\}$ is the edge $\mathbb{E}$,



(a) Inception Module Type-1: in the *initial offloading*, regard *layer block A-1* as one layer; in the *secondary offloading*, regard *layer block B-1* as one layer.

(b) Inception Module Type-2: in the *initial offloading*, regard *layer block A-2* as one layer; in the *secondary offloading*, regard *layer block B-2*, *layer block C-2*, and *layer block D-2* as one layer, respectively.

Fig. 2. Types of inception module.

and $\{M_{(2+b)}, M_{(3+b)}, \ldots, M_{(1+b+d)}\}$ is the cloud $\mathbb{C}$. Then, we define the server type $\boldsymbol{b} = \{b_0, b_1, b_2\}$, where b_0 represents the client, b_1 is the edge server, and b_2 is the cloud server.

The DNN set is denoted by $\mathbb{D} = \{D_1, D_2, \ldots, D_i, \ldots\}$, $i \in [1, \alpha]$, where the DNN model D_i is treated as a task, and α indicates the number of tasks. In the *Initial offloading*, each DNN task can be expressed as $D_i = \{D_{i1}, D_{i2}, \ldots, D_{ij}, \ldots\}$, $j \in [1, \beta_i]$, where we define the jth layer under the ith DNN as a subtask D_{ij}, and β_i refers to the number of subtasks under the task D_i. In the *Secondary offloading*, some subtasks D_{ij} are Inception Modules in the task D_i. We define the DNN layer under the Inception Module as $D_{ij^*}^{\upsilon w}$, where υ is the total number of branches in all Inception Modules, w represents the number of layers in each branch of the Inception Module, j^* indicates that the jth layer in the subtask D_{ij} is the Inception Module.

3.3 System Delay of Subtask

3.3.1 Execution Delay

In a local-edge-cloud collaborative environment, the subtask D_{ij} may be offloaded to the edge server, cloud server, or directly executed locally. We define an indicative function $\mathbf{1}_s$, where s represents the offloading location of the subtask D_{ij} in the environment $\mathbb{M}$. For instance, for the indicative function $\mathbf{1}_{s\in[2,1+b]}$, assuming that the actual offloading location of D_{ij} is M_s, when $s = 1$ or $s > 1 + b$, $\mathbf{1}_{s\in[2,1+b]} = 0$, otherwise, $\mathbf{1}_{s\in[2,1+b]} = 1$. After the input data required by the subtask D_{ij} is transmitted to the remote server, the subtask starts to execute, and its execution delay t_{ij}^1 can be defined as [28]:

$$t_{ij}^1 = \frac{w_{ij}}{Q_{ij}^L}\mathbf{1}_{s=1} + \frac{w_{ij}}{Q_{ij}^E}\mathbf{1}_{s\in[2,1+b]} + \frac{w_{ij}}{Q_{ij}^C}\mathbf{1}_{s\in[2+b,1+b+d]}, \tag{1}$$

where $Q_{ij}^L \in \{Q_1^L\}$, $Q_{ij}^E \in \{Q_1^E, \ldots, Q_n^E, \ldots, Q_b^E\}$, $Q_{ij}^C \in \{Q_1^C, \ldots, Q_m^C, \ldots, Q_d^C\}$. Q_{ij}^L, Q_{ij}^E and Q_{ij}^C indicate the computing power of the client, edge server and cloud server where the subtask $D_{ij}$is located, respectively. Q_1^L, Q_n^E and Q_m^C represent the computing power of the client L_1, edge server E_n and cloud server C_m, respectively. w_{ij} is the number of CPU cycles required to complete the subtask D_{ij}.

3.3.2 Transmission Delay

Since the DNN task D_i is a serial task, the subtasks are executed in sequence. We assume that the uplink data rate is the same as the downlink data rate. Based on Shannon's formula, the data transmission rate $v_u(\boldsymbol{a})$ under the server u can be calculated as follows [29]:

$$v_u(\boldsymbol{a}) = \omega \log_2\Big(1 + \frac{q_u g_{u,\rho}}{\varpi_0 + \sum_{\bar{n}\in\mathbb{U}\setminus\{u\}:a_{\bar{n}}=a_u} q_{\bar{n}} g_{\bar{n},\rho}}\Big), \tag{2}$$

where ω represents the bandwidth of the channel, q_u indicates the server transmission power, $g_{u,\rho}$ represents the channel gain between the server u and the base station ρ, ϖ_0 denotes the background noise power, $\mathbb{U} = \{1, 2, \ldots, U\}$ denotes the set of server u. For all servers, the decision profile $\boldsymbol{a} = \{a_1, a_2, \ldots, a_u, \ldots, a_U\}$, where a_u is the offloading decision under the server u. $a_u \in \{0\} \cup \mathbb{F}$, where $\{0\}$ indicates that data transmission will be suspended, $\mathbb{F} = \{1, 2, \ldots, F\}$ represents the set of wireless channels.

We consider the transmission delay caused by data transmission between different servers. The data transmission delay between subtasks D_{ij-1} and D_{ij} can be defined as [30]:

$$t_{ij}^2 = \frac{g_{ij}}{v_{ij}}, \tag{3}$$

where g_{ij} is the input data size of subtask D_{ij}. Then we calculate the transmission rate according to Eq. (2), let v_{ij} denote the data transmission rate between the servers where the subtask D_{ij-1} and the subtask D_{ij} are located respectively.

3.3.3 Queuing Time

The parallel pool represents the maximum number of subtasks that each remote server can run simultaneously. We assume that the number of pools of each server is limited, that is, computing resources are limited. Therefore, when offloading large-scale DNN models in parallel, it is necessary to consider the queuing time caused by the limited parallel pool. Let x_s^{pl} represent the number of parallel pools under the server M_s, x_s^{run} represents the number of subtasks running on the server M_s, and x_s^{st} represent the number of subtasks to be run under the server M_s. Obviously, we know that $x_s^{run} \leqslant x_s^{pl}$. In addition, if $x_s^{st} \leqslant x_s^{pl}$, the waiting time is 0. If $x_s^{st} > x_s^{pl}$, we calculate the time difference I_{ij} between the complete execution time of subtask being executed and the start execution time of subtask being queued. If $I_{ij} > 0$, indicating that the subtask needs to wait, and the waiting time $z_{ij} = I_{ij}$. If $I_{ij} \leqslant 0$, indicating that the subtask does not

need to wait, then the waiting time $z_{ij} = 0$. The queuing time t_{ij}^3 can be expressed as:

$$t_{ij}^3 = z_{ij}\mathbf{1}_{s=1} + z_{ij}\mathbf{1}_{s\in[2,1+b]} + z_{ij}\mathbf{1}_{s\in[2+b,1+b+d]}, \tag{4}$$

where $I_{ij} = t_{i'j'}^2 + t_{i'j'}^1 + z_{i'j'} - t_{ij}^2 + (\hat{t}_{i'j'} - \hat{t}_{ij})$, the waiting time of subtasks D_{ij} and $D_{i'j'}$ can be expressed as z_{ij} and $z_{i'j'}$, respectively, where $z_{ij} = \begin{cases} 0, & x_s^{st} \leqslant x_s^{pl} \\ \max\{0, I_{ij}\}, & x_s^{st} > x_s^{pl} \cdot t_{ij}^2 \end{cases}$ and $t_{i'j'}^2$ denote the data transmission delay of subtasks D_{ij} and $D_{i'j'}$, respectively. $t_{i'j'}^1$ is the execution delay of the subtask $D_{i'j'}$, $\hat{t}_{ij}$ and $\hat{t}_{i'j'}$ indicate the time to start transmitting the input data of the subtask D_{ij} and $D_{i'j'}$, respectively.

3.3.4 Total Delay

The total delay of the subtask D_{ij} can be expressed as follows:

$$T_{ij} = t_{ij}^1 + t_{ij}^2 + t_{ij}^3, \tag{5}$$

where T_{ij} is the sum of execution delay t_{ij}^1, transmission delay t_{ij}^2 and queueing time t_{ij}^3.

3.4 Energy Consumption of Subtask

3.4.1 Execution Energy Consumption

For clients and remote servers, the execution energy consumption can be expressed as follows [31]:

$$e_{ij}^1 = P_{ij}^L t_{ij}^1 \mathbf{1}_{s=1} + P_{ij}^0 t_{ij}^1 \mathbf{1}_{s\in[2,1+b]} + P_{ij}^0 t_{ij}^1 \mathbf{1}_{s\in[2+b,1+b+d]},$$

where P_{ij}^L denotes the power of the client when computing subtask D_{ij} locally, P_{ij}^0 represents the idle power of the client when the subtask D_{ij} is executed on the edge/cloud.

3.4.2 Transmission Energy Consumption

We consider the transmission energy consumption during the data transmission process, which can be calculated as [31]:

$$e_{ij}^2 = P_{ij}^t t_{ij}^2,$$

where P_{ij}^t is the transmission power from the client to the edge/cloud server where the subtask D_{ij} is located.

3.4.3 Total Energy Consumption

The total energy consumption of DNN subtask D_{ij} under each server is generated by data transmission and execution calculation. Therefore, the total energy consumption can be expressed as:

$$E_{ij} = e_{ij}^1 + e_{ij}^2. \tag{6}$$

3.5 Calculation Cost of Subtask

3.5.1 Execution Cost

The execution cost of the subtask D_{ij} can be obtained as follows [32]:

$$u_{ij}^1 = t_{ij}^1 q_{ij}^L \mathbf{1}_{s=1} + t_{ij}^1 q_{ij}^E \mathbf{1}_{s\in[2,1+b]} + t_{ij}^1 q_{ij}^C \mathbf{1}_{s\in[2+b,1+b+d]}, \tag{7}$$

where $q_{ij}^L \in \{q_1^L\}$, $q_{ij}^E \in \{q_1^E, \ldots, q_n^E, \ldots, q_b^E\}$, $q_{ij}^C \in \{q_1^C, \ldots, q_m^C, \ldots, q_d^C\}$. q_{ij}^L, q_{ij}^E and q_{ij}^C refers to the running cost per unit time of the client, edge server and cloud server where the subtask D_{ij} is located, respectively. q_1^L, q_n^E and q_m^C represent the running cost per unit time of the client L_1, edge server E_n and cloud server C_m, respectively.

3.5.2 Transmission Cost

We consider the cost generated by data transmission, and the transmission cost between subtasks D_{ij-1} and D_{ij} is as follows [33]:

$$u_{ij}^2 = g_{ij} \cdot y_{ij}. \tag{8}$$

where g_{ij} represents the transmission data from the subtask D_{ij-1} to the subtask D_{ij}, y_{ij} denotes the transmission cost per unit data between the servers where the subtask D_{ij-1} and the subtask D_{ij} are located respectively.

3.5.3 Total Cost

We consider that the total cost of subtask D_{ij} is generated by data transmission and task execution, so the total cost U_{ij} can be expressed as:

$$U_{ij} = u_{ij}^1 + u_{ij}^2. \tag{9}$$

3.6 Coupling Coordination and Node Balance

3.6.1 Coupling Coordination Degree

The coupling coordination degree is regarded as an important indicator to measure the coordination status of different indicators. We define the coupling coordination degree as a measurement indicator, which can more objectively and comprehensively measure the overall coordination level of "delay-energy-cost" system, reflecting the coordination effect of delay, energy consumption, and cost. The coupling coordination degree H_{ij}, the coupling degree C_{ij} and the comprehensive coordination indicator Y_{ij} of subtask D_{ij} [34], [35] can be expressed as follows, respectively:

$$H_{ij} = \sqrt{C_{ij} \times Y_{ij}}, \tag{10}$$

$$\begin{cases} C_{ij} = 3 \times \sqrt{[3]{\frac{T_{ij} \times E_{ij} \times U_{ij}}{(T_{ij}+E_{ij}+U_{ij})^3}}}, \\ Y_{ij} = \lambda_1 T_{ij} + \lambda_2 E_{ij} + \lambda_3 U_{ij}, \end{cases} \tag{11}$$

where $H_{ij} \in [0, 1]$, which is the higher the better. λ_1, λ_2 and λ_3 are weighting coefficients.

3.6.2 Node Balance Degree

In the real-world offloading scenario, the number of clients changes dynamically, and new users will continue to enter the local-edge-cloud collaborative environment. When the user enters the environment first has occupied all the computing resources of the server M_{s_0}, if the user enters the environment later wants to offload the subtasks to the server M_{s_0}, restrictions will occur. In order to improve the resource utilization of servers and achieve a balanced distribution of subtasks in the computing environment, the node balance is defined as:

$$Z_{ij} = \frac{1}{2}\sum_{s',s''=2}^{1+b} |B_{ij}^{s'} - B_{ij}^{s''}| + \frac{1}{2}\sum_{s',s''=2+b}^{1+b+d} |B_{ij}^{s'} - B_{ij}^{s''}|, \tag{12}$$

where $\frac{1}{2}\sum_{s',s''=2}^{1+b} |B_{ij}^{s'} - B_{ij}^{s''}|$ represents the node balance degree at the edge, $\frac{1}{2}\sum_{s',s''=2+b}^{1+b+d} |B_{ij}^{s'} - B_{ij}^{s''}|$ represents the node balance degree at the cloud, Z_{ij} indicates the node balance degree after offloading the subtask D_{ij} in the local-edge-cloud collaborative environment, $B_{ij}^{s'}$ and $B_{ij}^{s''}$ respectively denote the number of subtasks on servers $M_{s'}$ and $M_{s''}$ after the subtask D_{ij} is offloaded. In addition, since there is an absolute value in Eq. (12), $|B_{ij}^{s'} - B_{ij}^{s''}| = |B_{ij}^{s''} - B_{ij}^{s'}|$ is equal to the subtask difference calculated twice, so divide it by two.

3.7 Problem Formulation

In order to achieve "delay-energy-cost" optimization under the premise of multi-task parallel scheduling, DNN offloading in the local-edge-cloud collaborative environment can be regarded as an optimization problem. We are committed to finding the DNN offloading strategy with low delay, low energy consumption and low cost in a local-edge-cloud collaborative environment, while trying to optimize node balance degree and coupling coordination degree. In order to adapt to DNN offloading of chain type DNN model and topology type DNN model, we propose *two-step offloading strategy*, the details are described as follows:

3.7.1 Initial Offloading

In the first step of *two-step offloading strategy*, we take the Inception module as a whole and offload the overall DNN model. Then, we introduce a system utility $Q(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{x})$ defined as follows:

$$Q(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{x}) = \sum_i \sum_j \phi(\theta T_{ij} + \vartheta E_{ij} + \eta U_{ij}) - \varphi H_{ij} + \xi Z_{ij},$$

where $\boldsymbol{x} = \{x_s^{run} | s = 1, 2, \ldots, 1 + b + d\}$. $\boldsymbol{A}$ is the set of A_j^i, A_j^i denotes the offloading action of the subtask D_{ij}. $\boldsymbol{S}$ is the set of states S_j^i, S_j^i includes the number of CPU cycles required to complete the subtask D_{ij}, the transmission data between the subtasks D_{ij-1} and D_{ij} and the action information A_j^i. In addition, A_j^i can determine to which server the subtask D_{ij} will be offloaded, so that the server parameters can be obtained. A_{j-1}^i and A_j^i can determine the server, where the subtasks D_{ij-1} and D_{ij} are located, and then obtain the transmission rate between the two servers. For an environment with eleven servers, the offloading action of the subtask D_{ij} can be expressed as $A_j^i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, where each number represents a server.

Then, we formulate an optimization problem ($\mathcal{P}_1$) to minimize $Q(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{x})$ by jointly optimizing the offloading decision and the pool allocation, which is expressed as follows:

$$(\mathcal{P}_1): \quad Q^*(\boldsymbol{S}) = \min_{\boldsymbol{A},\boldsymbol{x}} Q(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{x}), \tag{13}$$

$$\text{s.t.}: \quad x_s^{run} \leqslant x_s^{pl}, \tag{14}$$

$$0 \leqslant \theta, \vartheta, \eta \leqslant 1, \tag{15}$$

$$0 \leqslant \phi, \varphi, \xi \leqslant 1, \tag{16}$$

$$\theta + \vartheta + \eta = 1, \tag{17}$$

$$\phi + \varphi + \xi = 1. \tag{18}$$

Considering the user's demand, we set the weights of delay, energy consumption, and cost to θ, ϑ, and η, respectively. In addition, we also set the weights of "delay-energy-cost" system score (SC), coupling coordination, and node balance to ϕ, φ, and ξ, respectively. For different task sizes, task numbers, and offloading environments, users usually have different offloading requirements:

- The choice of θ, ϑ, and η depends more on the user's offloading demands. For example, if the user urgently needs to obtain the DNN inference results, the importance of the delay far exceeds the energy consumption and cost, so choosing a larger delay weight θ is a better choice.
- The choice of ϕ, φ, and ξ depends more on environmental characteristics and task characteristics. For an offloading environment with sufficient computing resources, we can choose to ignore coupling coordination and node balance ($\varphi = \xi = 0$). For environments where computing resources are scarce, node balance degree and coupling coordination degree will be important, and φ and ξ can be set to higher values.

Although task optimization can be achieved objectively, in most cases, we still need to consider the subjective demands of users. The user comprehensively considers the size of the task, the number of tasks, and the offloading environment to set the corresponding weight, which is suitable for flexibly offloading tasks in different offloading environments.

3.7.2 Secondary Offloading

After *Initial Offloading*, we can know the offloading location of the input layer and output layer of the Inception module. On this basis, we divide and offload the Inception module. We need to note that the delay of the Inception module is the single branch with the longest delay, while the energy consumption and cost are the sums of all branches. On the basis of the *initial offloading*, we can obtain the system utility $Q(\boldsymbol{S'}, \boldsymbol{A'}, \boldsymbol{x})$ for the *secondary offloading*:

$$Q(\boldsymbol{S'}, \boldsymbol{A'}, \boldsymbol{x}) = \phi \sum_u \left\{ \max_{j^*} \sum_w \theta T_{ij^*}^{uw} \right\} + \sum_\upsilon \sum_w \phi\big(\vartheta E_{ij^*}^{\upsilon w} + \eta U_{ij^*}^{\upsilon w}\big) - \varphi H_{ij^*}^{\upsilon w} + \xi Z_{ij^*}^{\upsilon w}, \tag{19}$$

where $\boldsymbol{S'}$ denotes the set of states $S_{j'}^{iuw}$, $S_{j'}^{iuw}$ includes the number of CPU cycles required to complete the subtask $D_{ij^*}^{\upsilon w}$, the transmission data between the subtasks $D_{ij^*}^{\upsilon w-1}$ and $D_{ij^*}^{\upsilon w}$ and the action information. $\boldsymbol{x} = \{x_s^{run} | s = 1, 2, \ldots, 1 + b + d\}$. $\boldsymbol{A'}$ is the set of $A_{j^*}^{iuw}$, and $A_{j^*}^{iuw}$ refers to the offloading position of the subtask $D_{ij^*}^{uw}$. In addition, the calculation formulas of $T_{ij^*}^{uw}$, $E_{ij^*}^{uw}$, $U_{ij^*}^{uw}$, $H_{ij^*}^{uw}$, and $Z_{ij^*}^{uw}$ are the same as the calculation formulas of T_{ij}, E_{ij}, U_{ij}, H_{ij}, and Z_{ij}. u is the total number of Inception Modules, υ is the total number of branches in all Inception Modules, w represents the number

of layers in each branch of the Inception Module, j^* indicates that the jth layer in the subtask D_{ij} is the Inception Module.

$$Q^*(\boldsymbol{S'}) = \min_{\boldsymbol{A'},\boldsymbol{x}} Q(\boldsymbol{S'}, \boldsymbol{A'}, \boldsymbol{x}), \tag{20}$$

where the constrained conditions of $Q^*(\boldsymbol{S'})$ is the same as Eqs. (1) and (2).

4 DNN Offloading Using Deep Reinforcement Learning

Traditional algorithms usually perform continuous iterations to adjust the offloading strategy, but it is difficult to deal with high-complexity problems. DRL algorithm combines the perception ability of deep learning with the decision-making ability of reinforcement learning, and can directly learn control strategies from high-dimensional raw data. In order to solve the complexity problem, we propose an improved *DDPQN* algorithm based on the DRL algorithm, which can obtain an efficient DNN offloading strategy. In addition, in order to explain more concisely, we mainly discuss Eq. (3) in *initial offloading*. (Inception Module offloading in *secondary offloading* is the same as the *initial offloading* except for the difference of the reward function).

4.1 Modeling the DNN Offloading Process

In order to achieve low delay, low energy consumption, and low cost, we model the DNN offloading process as a Markov Decision Process (MDP). Our goal is to find an offloading function f to generate the optimal offloading strategy A^* for ($\mathcal{P}_1$), which can be expressed as:

$$f : \boldsymbol{S} \rightarrow \boldsymbol{A^*}, \tag{21}$$

where $\boldsymbol{S}$ represents the state, including the task calculation amount, transmission data, and action information. For instance, for α DNN tasks, each task D_i has β_i subtasks, and there are eleven servers in the offloading environment. Then $\boldsymbol{A^*} \in \{1,2,3,4,5,6,7,8,9,10,11\}^{\sum_{i=1}^{\alpha} \beta_i}$, the size of the offloading decision set $\{\boldsymbol{A}\}$ is $11^{\sum_{i=1}^{\alpha} \beta_i}$.

The designed *DDPQN* algorithm can learn the strategy function f step by step from experience. The details are described as follows:

4.1.1 Simplification of Offloading Problem

In the kth epoch, for state $\boldsymbol{S_k}$, a candidate offloading decision $\boldsymbol{A_t}$ will be generated.

$$f_k : \boldsymbol{S_k} \rightarrow \boldsymbol{A_t}, \tag{22}$$

where $\boldsymbol{A_t} \in \{1,2,3,4,5,6,7,8,9,10,11\}^{\sum_{i=1}^{\alpha} \beta_i}$ indicates the candidate offloading decision in the kth epoch.

We all know that $\mathcal{P}_1$ is a non-convex problem of mixed integer programming. If $\boldsymbol{A_t}$ is given, $\mathcal{P}_1$ will be simplified into a convex problem. At the same time, the original optimization problem ($\mathcal{P}_1$) becomes a pool allocation problem ($\mathcal{P}_2$), as shown in the following:

$$(\mathcal{P}_2): \quad Q^*(\boldsymbol{S}, \boldsymbol{A}) = \min_{\boldsymbol{x}} Q(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{x}) \tag{23}$$

$$\text{s.t.}: \quad \begin{aligned} & x_s^{run} \leqslant x_s^{pl}, \\ & 0 \leqslant \theta, \vartheta, \eta \leqslant 1, \end{aligned} \tag{24}$$

$$0 \leqslant \phi, \varphi, \xi \leqslant 1, \tag{25}$$

$$\theta + \vartheta + \eta = 1, \tag{26}$$

$$\phi + \varphi + \xi = 1. \tag{27}$$

The main requirement of ($\mathcal{P}_1$) is how to solve the problem of the DNN offloading strategy. For each input $\boldsymbol{S_k}$, a candidate offloading decision $\boldsymbol{A_t}$ is generated. Once the offloading decision $\boldsymbol{A_t}$ is given, the original optimization problem ($\mathcal{P}_1$) becomes a parallel pool allocation problem ($\mathcal{P}_2$). Then, select the lowest $Q^*(\boldsymbol{S_k}, \boldsymbol{A_t})$ offloading decision among all candidates, such as:

$$\boldsymbol{A_k^*} = \arg\min_{\boldsymbol{A_t} \in \boldsymbol{\Phi_k}} Q^*(\boldsymbol{S_k}, \boldsymbol{A_t}), \tag{28}$$

where $\boldsymbol{A_k^*}$ is the optimal offloading action in the kth epoch, $\boldsymbol{S_k}$ denotes the state. $\boldsymbol{\Phi_k} = \{\boldsymbol{A_t} | t = 1, 2, \ldots, B\}$ indicates the candidate action generated in the kth epoch, and B is the number of candidate offloading decisions.

4.1.2 Generation of Offloading Actions

DNN offloading in the local-edge-cloud collaborative environment is a dynamic system, in which each step of decision-making not only affects the current immediate rewards, but also the subsequent status and future rewards. In order to combine deep reinforcement learning with the DNN offloading problem, we model the DNN offloading process in the local-edge-cloud collaborative environment as a MDP and use the offloading environment as a training environment for the agent to perform reinforcement learning. Specifically, the basic elements for the MDP are defined as follows:

State: In order to comprehensively consider the characteristics between subtasks and servers, we define the state space at the jth step as:

$$S_j^i = [A_{j-1}^i, G_{j-1,j}^i, W_j^i, A_j^i, G_{j,j+1}^i, \ldots, G_{\beta_i-1,\beta_i}^i, W_{\beta_i}^i],$$

where A_j^i denotes the offloading action of the subtask D_{ij}, $G_{j-1,j}^i$ indicates the transmission data between subtasks D_{ij-1} and D_{ij}, and W_j^i denotes the number of CPU cycles required to complete the subtask D_{ij}. In addition, A_j^i can determine which server the subtask D_{ij} will be offloaded to, so that server parameters can be obtained. A_{j-1}^i and A_j^i can determine the server where the subtasks D_{ij-1} and D_{ij} are located, and then obtain the transmission rate between the two servers.

Action. The agent interacts with the environment and observes the state characteristics of the environment. We choose an offloading action of the subtask D_{ij} as $A_j^i \in \boldsymbol{A}$ and then offload it to the appropriate server.

Reward. The agent observes the environment and chooses action A_j^i according to the characteristic expression of the environment state S_j^i, and then the agent receives a reward

$R_{ij+1} \in R$ and enters a new state S^i_{j+1}. The objective is to minimize the overall result of the "delay-energy-cost" system in DNN offloading, given by Eq. (19). In order to achieve this goal, we define the reward score of the agent after each action as follows:

$$R_{ij} = \phi(\theta T_{ij} + \vartheta E_{ij} + \eta U_{ij}) - \varphi H_{ij} + \xi Z_{ij}.$$

4.1.3 Offloading Policy Update

Based on the traditional Deep Q-learning (DQN) algorithms, we design a *DDPQN* algorithm to optimize the evaluation metric of "delay-energy-cost" in the local-edge-cloud collaborative environment. We know that the algorithm saves a large amount of historical experience sample data, and each experience sample data is stored in a five-tuple $< s, a, r, s', T >$, meaning that the agent executes the action a in the state s, reaches the new state s', and obtains the corresponding reward r. Then the DQN algorithm according to the new state s' selects action a', where T is a Boolean value type, indicating whether the new state s' is a terminal state.

Neural Network Structure: We consider a more realistic situation where the size of the value function is independent of the action. For this reason, it is not necessary to estimate the Q value of each action. Combining the operation of the Dueling DQN [36], we divide the Q value update into two parts: the state value function $V(s)$ and the action advantage function $\Lambda(a)$. Different from the traditional fully connected layer, our network has two estimation streams, one is the state value estimation, and the other is the action advantage function estimation. The two together become the output of the Q-network.

$$Q(s, a, c_1, c_2) = V(s, c_2) + \Lambda(s, a, c_1) - \frac{1}{|\Lambda|}\sum_{a'} \Lambda(s, a', c_1),$$

where c_1 and c_2 are two parameters for estimating the stream network layer.

Experience Replay: For DQN random experience replay, we apply the prioritized replay mechanism from *PrioritizedReplay DQN* [37] to give different sample importance, improve the convergence speed, and avoid unnecessary redundant iterations, thereby optimizing the learning efficiency. We ensure that during the sampling process, the higher the priority, the higher the probability of being sampled, and the memory with the lowest priority also has a certain non-zero probability of being sampled. Specifically, we define the probability p^i_j as:

$$|R^i_{j+1} + \gamma^j_{j+1} \max_{a'} Q'(S^i_{j+1}, a') - Q(S^i_j, A^i_j)|^{\omega},$$

where ω indicates the hyperparameter that determines the shape of the distribution.

Parameter Update: Although the traditional DQN [38] algorithm can quickly make the Q value closer to the possible optimization goal, if we regard the maximum estimated value as an estimate of the maximum value of the true value, it will produce a positive maximization deviation. By separating the two steps of selecting the action corresponding to the Q value and evaluating the Q value corresponding to the action, the overestimation caused by the greedy algorithm is eliminated, and a more accurate Q value estimation is obtained, thereby making the learning more stable and reliable. The *DDPQN* algorithm is based on two neural networks [39] and uses gradient descent to update the parameters, so as to achieve the target value of parameter update L^i_j:

$$\left[R_{ij+1} + \gamma_{ij+1} Q'(S^i_{j+1}, \max_{a'} Q(S^i_{j+1}, a')) - Q(S^i_j, A^i_j)\right]^2,$$

where $\gamma_{ij+1} \in [0, 1]$ refers to the discount factor.

Algorithm 1. DDPQN Algorithm

Input: The parameters of the DNN model D_i and the local-edge-cloud collaborative environment M_s.
Output: Delay, energy consumption, cost, coupling coordination degree and node balance degree of subtask D_{ij}, DNN offloading strategy.

Preprocessing: The *DDPQN* algorithm is used to obtain the current optimal offloading decision and the historical offloading allocation plan, and then update the algorithm parameters.
Initialize DNN parameters under DDPQN.
Initialize empty memory pool $\boldsymbol{X}$.
Initialize state $\boldsymbol{S}$.
for *episode* $k = 1 \rightarrow N$ **do**
  for *task* $i = 1 \rightarrow \alpha$ **do**
    for *moment* $j = 1 \rightarrow \beta_i$ **do**
      Input state S^i_j to Q network to get Q values of all actions.
      Select action A^i_j using the ϵ-greedy.
      Execute action A^i_j in state S^i_j.
      Get new status S^i_{j+1}, reward R_{ij}.
      Put $\{S^i_j, A^i_j, R^i_j, S^i_{j+1}\}$ to memory pool $\boldsymbol{X}$.
      Sampling according to prioritized replay.
      $Q(s, a, c_1, c_2) = V(s, c_2) + \Lambda(s, a, c_1) - \frac{1}{|\Lambda|}\sum_{a'} \Lambda(s, a', c_1)$.
      if *terminate* **then**
        $z^i_j = R_{ij+1} + \gamma Q'(S^i_{j+1}, \max_a Q(S^i_{j+1}, a'))$.
      else
        $z^i_j = R_{ij+1}$.
      Update model parameters of loss function:
      $[z^i_j - Q(S^i_j, A^i_j)]^2$.
      Reset $Q' = Q$ per ς step.
final
return the optimal offloading decision

4.2 DDPQN Algorithm

We present the overall design of the *DDPQN* algorithm as shown in Algorithm 1, which mainly adapts to the state of the environment through the deep learning, and then makes reasonable decisions in each state based on reinforcement learning, so as to select reasonable actions and determine the offloading location of each subtask. At the same time, every decision will get feedback rewards from the environment. This value is used to guide the agent's learning, so that the agent can explore the direction of reward maximization, thereby optimizing the offloading goal.

In order to achieve efficient offloading of DNN tasks, we optimize the parameter update, experience replay, and neural network structure based on the original DQN algorithm to improve the offloading efficiency. In the entire model structure, we divide the *DDPQN* algorithm into two parts:

TABLE 1
Symbols and Definitions

Symbols	Definitions
$\mathbb{L}$	Client
$\mathbb{E}$	Edge
$\mathbb{C}$	Cloud
$\mathbb{M}$	Local-edge-cloud collaborative environment
$\mathbb{D}$	DNN model collection
D_{ij}	The jth layer under the ith DNN
t_{ij}^1	Execution delay
t_{ij}^2	Transmission delay
t_{ij}^3	Queuing Time
T_{ij}	Total Delay
e_{ij}^1	Execution energy consumption
e_{ij}^2	Transmission energy consumption
E_{ij}	Total energy consumption
u_{ij}^1	Execution cost
u_{ij}^2	Transmission cost
U_{ij}	Total cost
H_{ij}	Coupling coordination degree
Z_{ij}	Node balance degree

TABLE 2
Transmission Parameters Between Servers

b_i	b_j	bandwidth (MB/s)	cost ($/GB)	P(W)
b_2	b_2	5	0.4	$\times$
b_2	b_1	0.5	0.8	$\times$
b_0	b_2	0.5	0.8	0.2
b_1	b_1	10	0.16	$\times$
b_0	b_1	10	0.16	0.2

TABLE 3
Server Parameters

b_i	cost/hour ($)	CP (GHz)	P (W)
b_0	0	2.3	70
b_1	$2.10 \sim 2.43$	$4.2 \sim 18.3$	10
b_2	$0.225 \sim 1.80$	$40 \sim 120$	10

the state value function and the action advantage function, and the value of Q is updated accordingly. DQN stores the next state S_{j+1}^i and reward R_{j+1}^i, and stores quaternion groups $(S_j^i, A_j^i, R_j^i, S_{j+1}^i)$ as an experience in the memory pool, after update the DQN, the experience in the memory pool will be selected according to the prioritized replay mechanism, which can ensure effective learning of previous experience and avoid the limitations brought by continuous experience. In addition, the *DDPQN* algorithm uses another network to generate the Q value in the training process. The network structure is the same as the training neural network and Q is kept consistent. After iterations, the parameters of Q are copied to the target neural network Q'. Therefore, by maintaining the difference of the two network parameters for a period of time, the difference between the current Q value and the Q' value is used to calculate the loss function, and then stochastic gradient descent can be used to reversely update parameters of the network.

5 Performance Evaluation

In this section, we evaluate DNN offloading performance based on the *DDPQN* algorithm.

5.1 Experimental Setup

We first establish a local-edge-cloud collaborative environment $\mathbb{M} = \{M_1, M_2, \ldots, M_{11}\}$, where the servers $\{M_1\}$ belong to the client, the server $\{M_2, M_3, M_4, M_5, M_6\}$ belong to the edge, and the servers $\{M_7, M_8, M_9, M_{10}, M_{11}\}$ belong to the cloud. We define the weights of delay, energy and cost as $\theta = 0.5$, $\vartheta = 0.3$ and $\eta = 0.2$, the weights of the "delay-energy-cost" system score, the coupling coordination degree, and the node balance degree are $\phi = 0.6$, $\varphi = 0.2$, and $\xi = 0.2$, respectively. $\lambda_1 = \lambda_2 = 0.33$ and $\lambda_3 = 0.34$.

We conduct experiments using four different DNNs, i.e., GoogleNet [40], ResNet [41], AlexNet [42] and VGG [43]. The basic structure, computational amount, and transmission data of DNN model come from file[1]. The environment parameter settings [32], [44], [45] can be seen in Tables 2 and 3, where b_0, b_1 and b_2 denote the client, the edge server and the cloud server, respectively. This paper considers a fully connected DNN composed of one input layer and two hidden layers. The number of neurons in the two hidden layers is 50 and 20, respectively. Then create a linear fully connected layer of advantage function and value function, the output number of the advantage layer is 11, and the output number of the value layer is 1.

To reveal the advantages of the *DDPQN* algorithm in solving the DNN offloading problem, we compare it with commonly used computation offloading algorithms listed as follows:

- *DQN*: On the basis of Q-learning, an experience replay mechanism is introduced, and an offline and online two-layer neural network is established to improve training efficiency;
- *Dueling DQN*: The algorithm is improved by optimizing the structure of the neural network. Its network has two estimation streams, which estimate the state value and the action advantage function respectively;
- *Double DQN*: After training, the *Double DQN* makes the current Q value infinitely close to the Q_{target} value, so that the error between the two tends to be stable and close to 0;
- *PrioritizedReplay DQN*: The algorithm uses the prioritized replay mechanism to give samples different importance, thereby speeding up convergence and making learning more effective.
- *DoublePr DQN*: On the basis of *Double DQN*, a prioritized replay mechanism is added to speed up the convergence efficiency of the algorithm by reducing overestimation;
- *DuelingPr DQN*: Combine *Dueling DQN* with prioritized replay mechanism to improve high information utilization and algorithm performance;
- *Double Dueling DQN*: Combine *Dueling DQN* with *Double DQN* to get more useful information while avoiding overestimation of value.

1. https://github.com/LinBin403/dataset-for-our-research

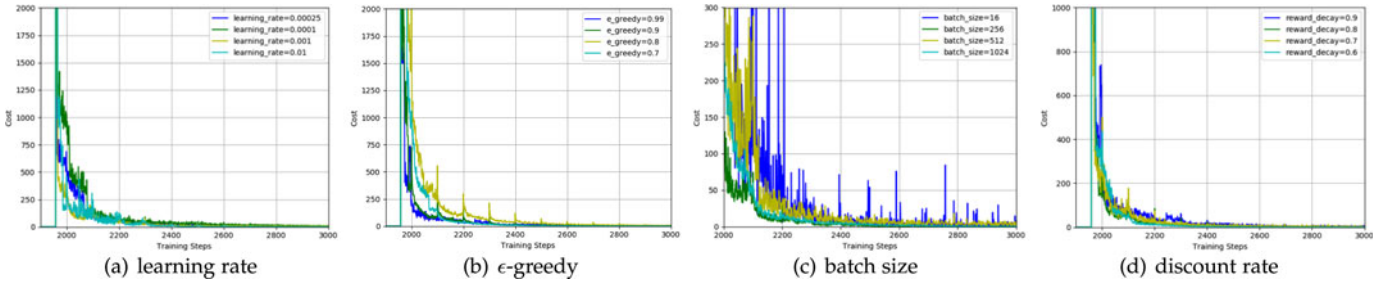


Fig. 3. Convergence performance under different parameters.

- *Greedy* algorithm: This is a common method to find the optimal offloading decision, which generally divides the solution process into several steps, but each step applies the greedy principle to select the best choice in the current state and hopes to stack the final results together.
- *DDPQN*: Based on traditional DQN, we have made relevant improvements in experience replay, neural network structure, and parameter update.

We consider the performance of the *DDPQN* algorithm in different computing environments, such as the local-edge-cloud collaborative environment (L-E-C), edge computing environment (L-E), and cloud computing environment (L-C). In addition, we record $\mathbf{A}$ as the "delay-energy-cost" system score, $\mathbf{B}$ as the coupling coordination degree, and $\mathbf{X}$ as the node balance degree. For example, $\mathbf{A}-\mathbf{B}-\mathbf{X}$ represents when the *DDPQN* algorithm trains the "delay-energy-cost" system, it considers the coupling coordination degree and node balance degree. $\mathbf{A}-\mathbf{X}$ represents when the *DDPQN* algorithm trains the "delay-energy-cost" system, it only considers the node balance degree.

5.2 Convergence Performance

We show the convergence performance of the *DDPQN* algorithm under different hyperparameters. In Fig. 3, the abscissa is the training step, and the ordinate is the loss of the neural network.

Fig. 3a shows the convergence efficiency of the *DDPQN* algorithm under different learning rates. When the learning rate is too high or too low, a good convergence effect cannot be obtained. When the learning rate is 0.001, we can obtain the best convergence effect. Fig. 3b shows the convergence effect of the algorithm under different ϵ-greedy. We know the explore and exploit are balanced by setting ϵ-greedy. The larger the value of ϵ-greedy, the more inclined to choose to maximize the current moment. For the action of expected profit, it is found that when ϵ-greedy is larger, the convergence effect of the algorithm is better, so we determine ϵ-greedy=0.9 as the parameter used in subsequent experiments. Fig. 3c shows the effect of sample batch size on convergence performance. Obviously, smaller or larger sample batches will bring poorer convergence effects. Thus, we set the batch value as 256 in subsequent experiments. In Fig. 3d, we can see the algorithm convergence at different discount rates. The discount rate λ determines the present value of future earnings, when the discount rate approaches 0, the agent will only maximize the current rewards more. In this experiment, if the discount rate is small, it will lead to poor convergence, so the discount rate is set as $\lambda = 0.8$.

In addition, since the *DDPQN* algorithm adds the prioritized replay mechanism, the iteration starts from the time when the first reward was originally obtained. We set the network parameters to the target network every 100 steps, and there will be fluctuations every 100 steps. It is mainly due to the parameter freezing mechanism, and this operation will not affect the convergence of the model.

5.3 Convergence and Generalization

In this section, we discussed the advantages of the *DDPQN* algorithm in convergence and generalization.

Figs. 4 compares the convergence of the *DDPQN* algorithm and the existing DRL algorithm in a local-edge-cloud collaborative environment. We find that the *DDPQN* algorithm has better convergence performance. Besides, since *DDPQN*, *PrioritizedReplay DQN*, *DoublePr DQN*, and *DuelingPr DQN* consider the prioritized replay mechanism, iterate after the first reward is completed.

Fig. 5 shows the performance of the *DDPQN* algorithm and the existing DRL algorithm under various DNN models with different structures. First, We find that the *DDPQN* algorithm can efficiently converge to an excellent average reward value whether it is in a chain DNN model with a relatively simple structure such as the AlexNet or in a complex topological DNN model such as the GoogleNet and the ResNet. Second, compared with the existing DRL algorithm, the *DDPQN* algorithm achieves the optimal average reward value under various DNN models with different structures, such as AlexNet, GoogleNet, and ResNet.

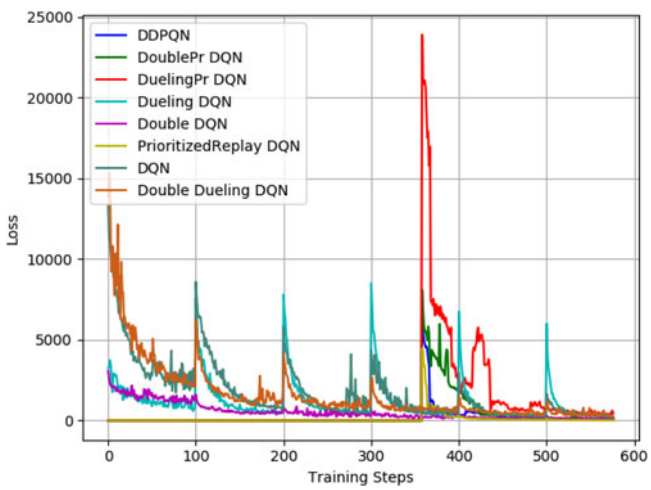


Fig. 4. Convergence performance under various algorithms.

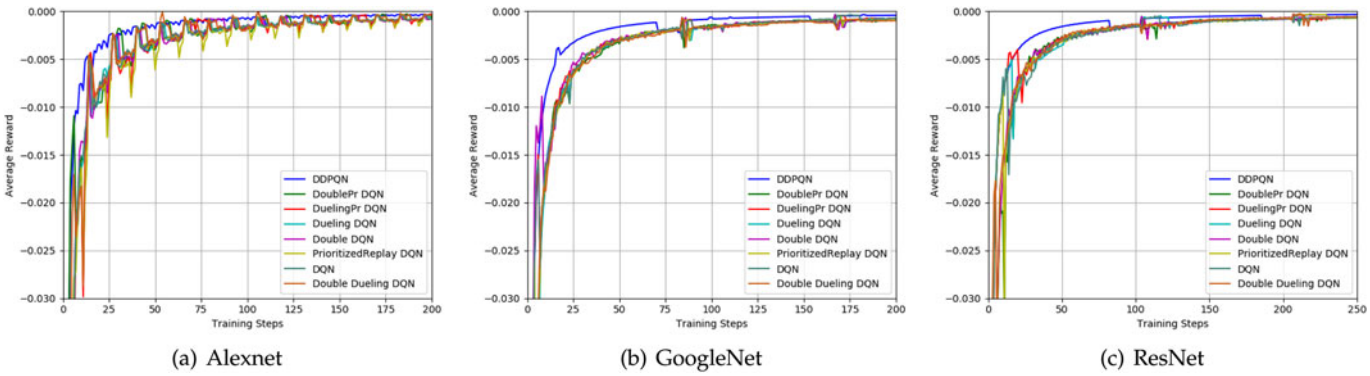


Fig. 5. The performance of the *DDPQN* algorithm under different types of DNN models.

In summary, we can ensure that the *DDPQN* algorithm is suitable for realizing efficient partitioning and offloading of various DNN models in the local-edge-cloud collaborative environment.

5.4 The Impact of the Number of Tasks

In this part, we compare the performance of *DDPQN*, *DQN* and *greedy* algorithms under different numbers of tasks.

It can be seen from Fig. 6 that the delay, energy consumption, and cost of the above three algorithms all increase linearly with the increase of the number of tasks. Since delay is a crucial factor in the task offloading process, we set the highest weight for the delay in the experiment. It can be observed from Fig. 6a that the *DDPQN* algorithm can achieve the lowest delay, and its delay growth trend tends to be slower as the amount of tasks increases. Obviously, the *DDPQN* algorithm is a more weight-sensitive algorithm and is more suitable for multi-index comprehensive offloading. In Fig. 6b, the energy consumption of the *DDPQN* algorithm is the highest. Due to the heavy weight of the delay indicator, for DNN subtasks with a large amount of calculation, in order to pursue the lower delay, the DNN subtasks are usually offloaded to the edge server, which generates high execution energy consumption, resulting in high total energy consumption. In Fig. 6c, the cost of the *DDPQN* algorithm is close to that of the *greedy* algorithm, based on Eqs. (7) and (8), which is mainly realized by low delay. All in all, the *DDPQN* algorithm is suitable for large-scale DNN offloading.

5.5 The Impact of Weights

This section first introduces the specific performance of delay, energy consumption, and cost under different weights. Then we discuss the performance of the "delay-energy-cost" system score (SC) (i.e., $\theta T_{ij} + \vartheta E_{ij} + \eta U_{ij}$), coupling coordination, and node balance under different weights.

In Fig. 7a, the ordinate adopts the standardized value of delay, energy consumption, and cost. The legend indicates the weight ratio of delay, energy consumption, and cost, that is, "$\theta : \vartheta : \eta$". In the combination of various types of $\theta : \vartheta : \eta$ ("$7:2:1$", "$1:7:2$", and "$2:1:7$"), the weight ratios of delay, energy consumption, and cost are "$7:1:2$", "$2:7:1$", and "$1:2:7$", respectively. We find that the greater the weight, the higher the degree of indicator optimization, and the smaller the weight, the lower the degree of indicator optimization. This means that delay, energy consumption, and cost are closely related to the weight values. In addition, we know that delay, energy consumption, and cost are highly correlated, and the level of one indicator will have an impact on other indicators. For example, the weight ratio of cost is "$1:2:7$", where the corresponding cost value of "1" is less than the cost value corresponding to "2", the reasons are as follows: First,the weight values of "1" and "2" are too small, so the influence on indicator optimization is too low. Second, for "1", the proportion of delay in "$\theta : \vartheta : \eta = 7:2:1$" is too large, so the delay is low, based on Eqs. (7) and (8), therefore the cost is low; on the contrary, for "2", the proportion of delay in "$\theta : \vartheta : \eta = 1:7:2$" is too small, so the cost is very large.

In Fig. 7b, the ordinate adopts the standardized value of SC, coupling coordination, and node balance. The legend indicates the weight ratio of SC, coupling coordination, and node balance, that is, "$\phi : \varphi : \xi$". In the combination of various types of $\theta : \vartheta : \eta$ ("$2:3:5$", "$6:2:2$" and "$8:1:1$"), the weight ratios of SC, coupling coordination, and node

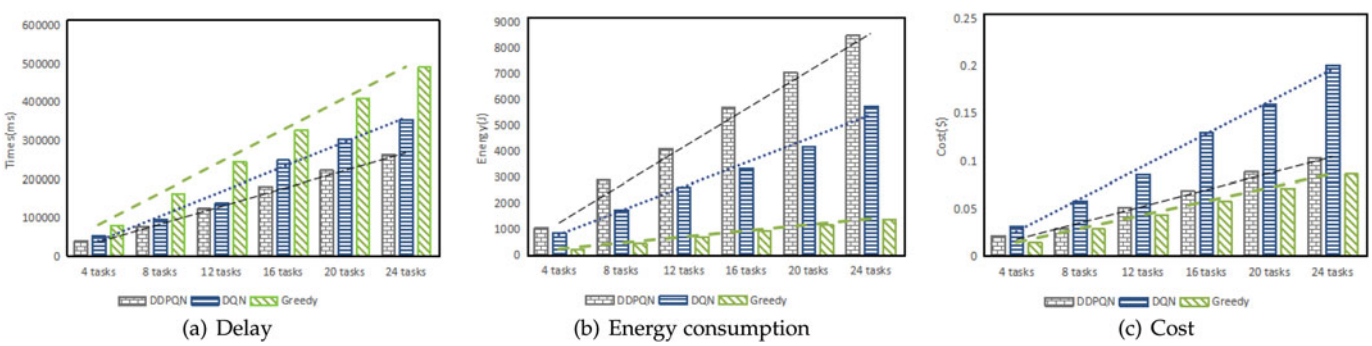
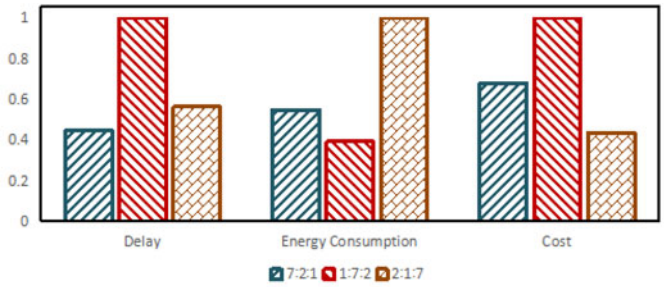
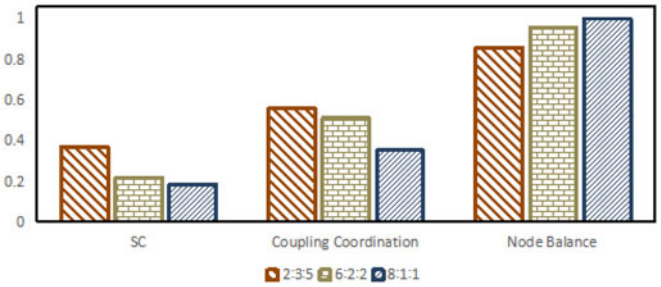


Fig. 6. The impact of the number of tasks on different algorithms.



(a) The impact of weights on delay, energy, and cost.



(b) The impact of weights on "delay-energy-cost" system score, coupling coordination, and node balance.

Fig. 7. The effect of weights on the optimization function.

balance are "$2:6:8$", "$3:2:1$", and "$5:2:1$", respectively. Obviously, as the weight increases, the SC shows a downward trend. In addition, we find that the SC and node balance are negatively correlated with the weight, while coupling coordination is positively correlated with the weight. In other words, SC, coupling coordination, and node balance are closely related to the weight.

5.6 The Impact of Coupling Coordination and Node Balance

In this part, we show the performance of coupling coordination and node balance under different DNN models.

Fig. 8 shows the coupling coordination degree of the DNN layer under different combinations of indicators, where the abscissa represents the DNN layers, and the ordinate represents the coupling coordination degree. Figs. 8a, 8c, and 8e show the coupling coordination performance of the "delay-energy-cost" system after adding the coupling coordination indicator **B**. It is found that the image area of **A-B** is significantly better than that of **A**. In other words, after adding **B**, **A-B** can achieve better coupling coordination performance than **A**. It can avoid the situation that the system performance is very good, but one indicator is the best and another indicator is extremely poor. Figs. 8b, 8d, and 8f show the coupling coordination performance of the "delay-energy-cost" system after adding the coupling coordination indicator **B** and the node balance indicator **X** at the same time. Obviously, by adding **B** and **X**, the coupling coordination performance of the **A-B-X** is better than **A-B**. This indicates that the node balance indicator has a beneficial effect on the coupling coordination indicator.

Fig. 9 shows the total coupling coordination degree of the DNN layer under **A-B-X**, **A-B**, **A-X**, and **A**. We can see that after adding **B**, the coupling coordination performance of the system has been greatly improved. In addition, we observe that the node balance indicator also has a positive impact on coupling coordination, and when the system adds the node balance indicator and the coupling coordination indicator, the coupling coordination performance of **A-B-X** is significantly better than that of **A**, **A-X** and **A-B**. Compared with **A**, the coupling coordination degree of **A-B-X** under GoogleNet, ResNet, AlexNet, and VGG is optimized by 44.29%, 59.13%, 117.53%, and 84.08%, respectively. This shows that in the local-edge-cloud collaborative environment, adding the node balance indicator and coupling coordination indicator will further improve the performance of coupling coordination.

Fig. 10 shows the impact of node balance indicators on subtask distribution status of GoogleNet, ResNet, AlexNet and VGG under the combination of **A-B-X**, **A-X**, and **A**. Each histogram consists of two parts, where the lower part represents the node balance degrees in the edge, and the upper part represents the node balance degrees in the cloud. We can clearly see that after adding **X**, the node balance value of **A-X** is significantly better than **A**. In addition, after adding coupling coordination indicator and node balance indicator, the node balance value of **A-B-X** achieves the optimal performance. Compared with **A**, the node balance degree of **A-B-X** under GoogleNet, ResNet, AlexNet, and VGG is optimized by 11.82%, 9.77%, 13.70%, and 7.51%, respectively. That is to say, the coupling coordination degree and the node balance degree have a positive mutual gain effect, which can effectively obtain a better DNN offloading strategy.

5.7 The Impact of Coordination and Balance on the System

Fig. 11 shows the impact of coupling coordination degree and node balance degree on delay, energy consumption, and cost under different types of DNN models. Although increasing the coupling coordination indicator or node balance indicator will optimize the system coordination and subtask distribution balance, it will inevitably bring the burden on delay, energy consumption, and cost. As shown in Figs. 11a, 11b, and 11c, we obtain higher coupling coordination degree and node balance degree by sacrificing less delay, energy consumption, or cost, which is an ideal situation. It can avoid the situation where the system evaluation result of "delay-energy-cost" is excellent, but the difference of internal indicator optimization is too large. It can also avoid premature saturation of some servers because users who enter the environment first greedily occupy server resources, making it difficult for users who enter the environment later to obtain a reasonable allocation of computing resources in the local-edge-cloud collaborative environment.

5.8 The Impact of Various Environments

We considered the delay, energy consumption, and cost performance of GoogleNet, ResNet, AlexNet, and VGG in a local-edge-cloud collaborative environment, edge computing environment, and cloud computing environment, respectively.

Fig. 12a shows the delay performance of the four DNN tasks in the "delay-energy-cost" system under different computing environments. We find that the delay

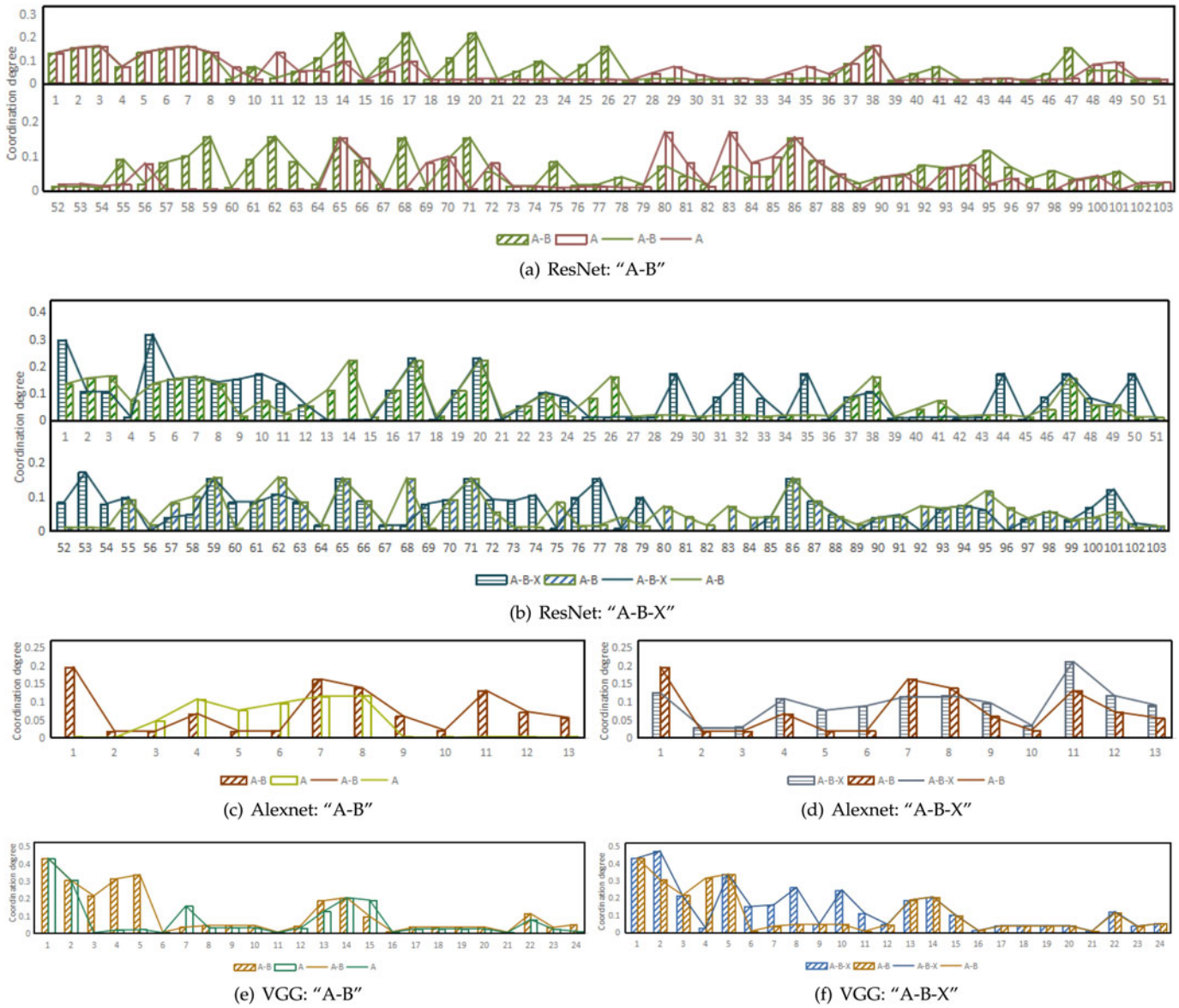


Fig. 8. The performance of coupling coordination degree under different indicator combinations.

performance is the best in the local-edge-cloud collaborative environment and the worst in the cloud computing environment. This is because the cloud is too far away from the data source, resulting in a higher transmission delay in the cloud computing environment. In the edge computing environment, although the edge server is close to the data center, the data transmission delay is low, but due to its limited

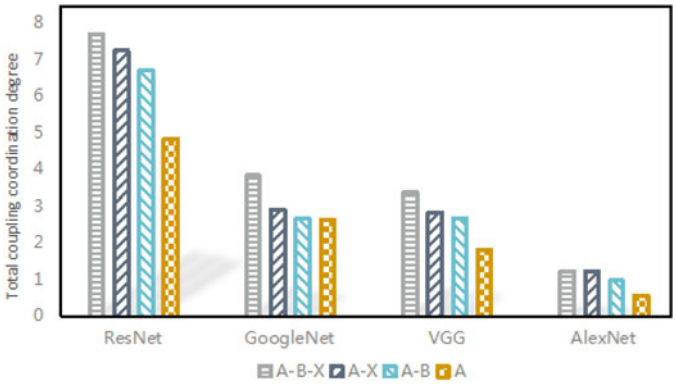


Fig. 9. The performance of the total coupling coordination degree of different indicator combinations under GoogleNet, ResNet, AlexNet, and VGG.

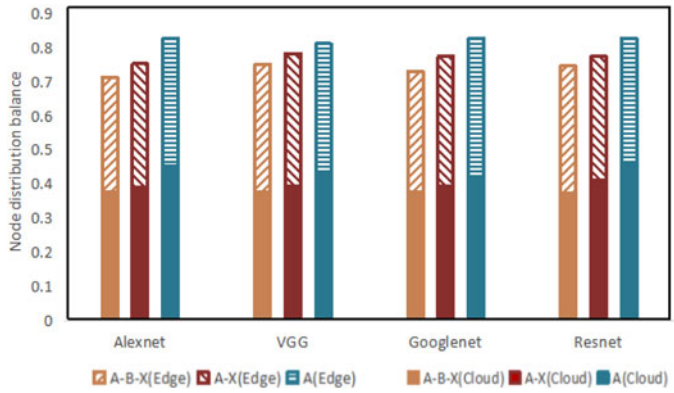


Fig. 10. The performance of the node balance degree of different indicator combinations under GoogleNet, ResNet, AlexNet, and VGG.

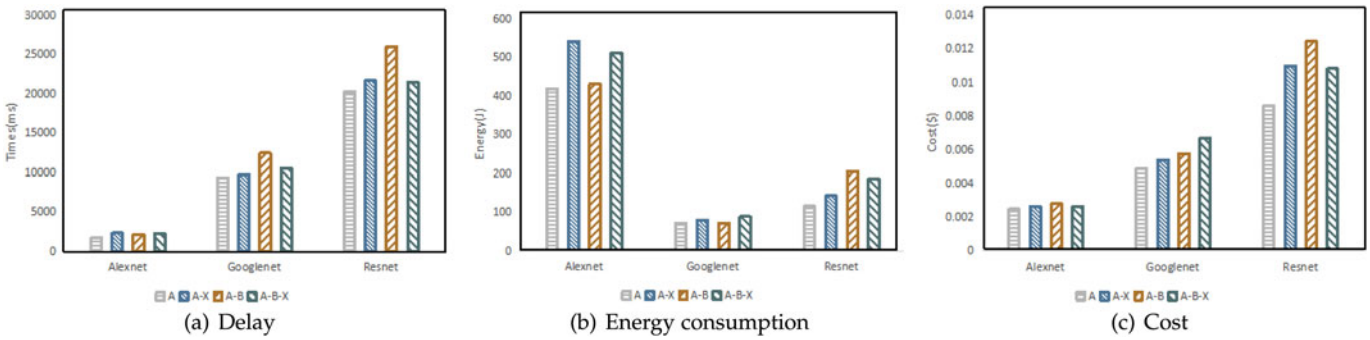


Fig. 11. The impact of coupling coordination degree and node balance degree on delay, energy, and cost.

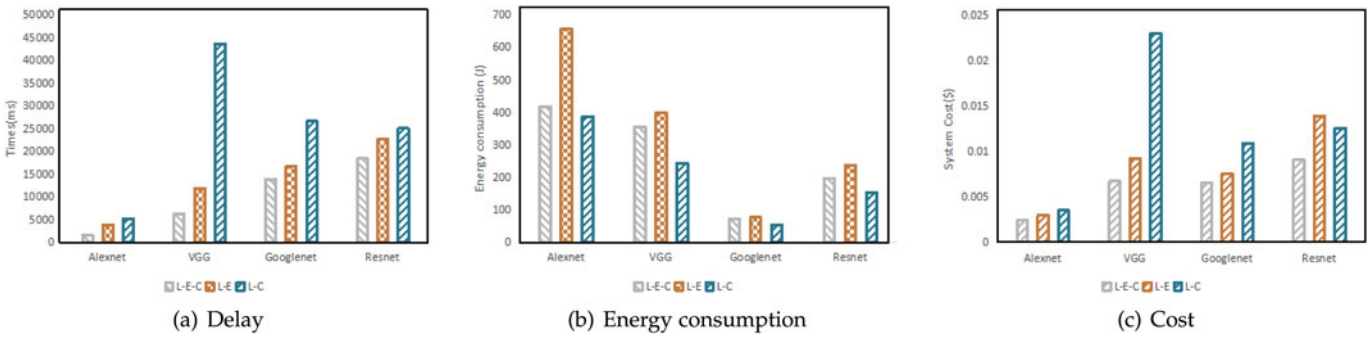


Fig. 12. Comparison of delay, energy consumption and cost under different computing environments.

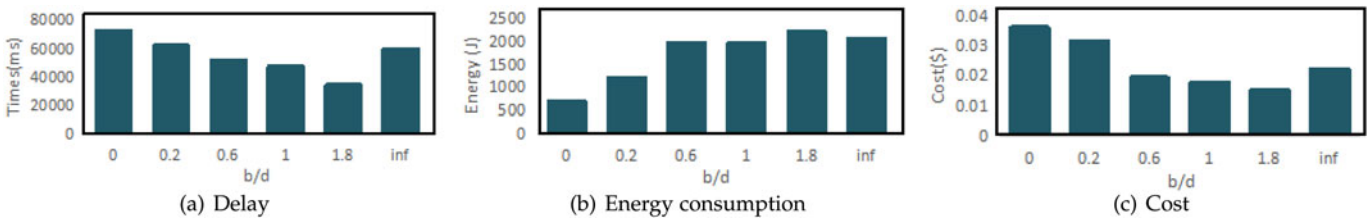


Fig. 13. The performance of delay, energy consumption, and cost in an offloading environment with various server combinations. b and d are the numbers of edge servers and cloud servers, respectively, b/d denotes the ratio between the edge servers and cloud servers, and inf denotes the number of cloud servers is zero.

computing power, it is difficult to support large-scale DNN tasks. Therefore, offloading subtasks in a local-edge-cloud collaborative environment that combines the advantages of edge servers and cloud servers can achieve the best delay performance.

Fig. 12b shows the performance of energy consumption in the "delay-energy-cost" system. We find that there is a trade-off between delay and energy consumption. The total delay in the cloud computing environment is higher than that in the edge computing environment. Compared with edge servers, cloud servers usually have higher computing power and lower execution delay. For the DNN tasks with a large amount of calculation, considering the lower execution energy consumption in the cloud computing environment, lower total energy consumption can be obtained in the cloud computing environment compared to the edge computing environment. In a local-edge-cloud collaborative environment, combining the respective advantages of the cloud and the edge can optimize delay and energy consumption at the same time.

Fig. 12c shows the cost performance of GoogleNet, ResNet, AlexNet, and VGG in the "delay-energy-cost" system. We hope to reduce the execution cost and transmission cost of the task on the basis of achieving low delay and low energy consumption. As shown in Fig. 12c, compared to the edge computing environment and the cloud computing environment, the system cost under the local-edge-cloud collaborative environment is lower. We find that cloud transmission costs are high, but execution costs are low; edge transmission costs are low, but execution costs are high. The local-edge-cloud collaborative environment can combine the advantages of both edge and cloud, which is more suitable for resource offloading.

5.9 Performance Under Different Server Combinations

In this section, we will offload the DNN model in a local-edge-cloud collaborative environment with different server combinations.

In Figs. 13a and 13c, we find that as the number of edge servers increases, the delay and cost become lower. However, when the number of cloud servers is 0 and the edge servers are limited, the delay and cost will be higher due to the

limitation of computing resources. From Figs. 13a and 13b, we can clearly see that for DNN tasks with a large amount of calculation, owing to the excessive weight of the delay, as the number of edge servers increases, the subtasks will tend to be offloaded to the edge servers to obtain the lower delay. At this time, the execution energy consumption gradually increases, so the total energy consumption also increases.

Furthermore, comparing the server ratio $b/d \in \{0, 0.2, \inf\}$, we find that when the server ratio $b/d \in \{0.6, 1, 1.8\}$, by appropriately increasing the edge servers, better offloading performance can be achieved.

6 Conclusion

Based on the abundant resources of the cloud and the low delay advantages of the edge, this paper studies the optimization of DNN partitioning and offloading in a local-edge-cloud collaborative environment. In order to solve the problem of uncoordinated optimization of multiple indicators and unreasonable allocation of computing resources, we introduce coupling coordination indicator and node balance indicator to achieve high-quality DNN partitioning and offloading. In this paper, the proposed *DDPQN* algorithm can generate the optimal DNN task allocation with low energy consumption, low delay, and low cost in a local-edge-cloud collaborative environment, which effectively improves the QoS. The experimental results show that the *DDPQN* algorithm has better performance in optimizing DNN offloading compared with the existing DRL algorithms. In future work, we will consider a more realistic scenario, that is, implementing dynamic offloading in a local-edge-cloud collaborative environment. Besides, we will combine the Graph Neural Network (GNN) for DNN offloading to further accelerate DNN inference.

References

[1] H. Wu, "Analysis of offloading decision making in mobile cloud computing," Ph.D. dissertation, Dept. Math. Comput. Sci., Freie Universität Berlin, Berlin, Germany, 2015.

[2] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.

[3] H. Q. Le, H. Al-Shatri, and A. Klein, "Efficient resource allocation in mobile-edge computation offloading: Completion time minimization," in *Proc. IEEE Int. Symp. Inf. Theory*, 2017, pp. 2513–2517.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[5] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.

[6] Q. Li, S. Wang, A. Zhou, X. Ma, fangchun yang, and A. X. Liu, "QoS driven task offloading with statistical guarantee in mobile edge computing," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2020.3004225.

[7] Q. Luo, C. Li, T. Luan, and W. Shi, "Minimizing the delay and cost of computation offloading for vehicular edge computing," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2021.3064579.

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

[9] D. Loghin, L. Ramapantulu, and Y. M. Teo, "Towards analyzing the performance of hybrid edge-cloud processing," in *Proc. IEEE Int. Conf. Edge Comput.*, 2019, pp. 87–94.

[10] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[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 *Proc. ACM Symp. Cloud Comput.*, 2018, Art. no. 401–411.

[12] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans Wireless Commun*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[13] B. Qi, M. Wu, and L. Zhang, "A DNN-based object detection system on mobile cloud computing," in *Proc. 17th Int. Symp. Commun. Inf. Technologies*, 2017, pp. 1–6.

[14] 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 Commun.*, vol. 27, no. 1, pp. 92–99, Feb. 2020.

[15] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1423–1431.

[16] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGPLAN Not.*, vol. 52, no. 4, pp. 615–629, Apr. 2017.

[17] H. Wang, G. Cai, Z. Huang, and F. Dong, "ADDA: Adaptive distributed DNN inference acceleration in edge computing environment," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst.*, 2019, pp. 438–445.

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

[19] W. Ju, D. Yuan, W. Bao, L. Ge, and B. B. Zhou, "Deepsave: Saving DNN inference during handovers on the edge," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, 2019, pp. 166–178.

[20] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *Proc. Conf. Comput. Commun.*, 2020, pp. 854–863.

[21] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 328–339.

[22] 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 Netw.*, vol. 34, no. 2, pp. 254–261, Mar. 2020.

[23] Z. Chen, J. Hu, X. Chen, J. Hu, and G. Min, "Computation offloading and task scheduling for DNN-based applications in cloud-edge computing," *IEEE Access*, vol. 8, pp. 115 537–115 547, 2020.

[24] C. Ding, A. Zhou, Y. Liu, R. Chang, and S. Wang, "A cloud-edge collaboration framework for cognitive service," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2020.2997008.

[25] R. G. Pachecom and R. S. Couto, "Inference time optimization using BranchyNet partitioning," in *Proc. IEEE Symp. Comput. Commun.*, 2020, pp. 1–6.

[26] 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, Jan. 2021, Art. no. 229.

[27] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city internet of things," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8099–8110, Sep. 2020.

[28] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Deep reinforcement learning for stochastic computation offloading in digital twin networks," *IEEE Trans. Ind. Inform.*, vol. 17, no. 7, pp. 4968–4977, Jul. 2021.

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

[30] L. Huang, X. Feng, A. Feng, Y. Huang, and P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, pp. 1–8, Nov. 2018, doi: 10.1007/s11036-018-1177-x.

[31] X. Xu *et al.*, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.

[32] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven off-loading for DNN-based applications over cloud, edge, and end devices," *IEEE Trans. Ind. Inform.*, vol. 16, no. 8, pp. 5456–5466, Aug. 2020.

[33] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Mar. 2022.
[34] F. Dong and W. Li, "Research on the coupling coordination degree of "upstream-midstream-downstream" of china's wind power industry chain," *J. Cleaner Prod.*, vol. 283, Feb. 2021, Art. no. 124633.
[35] D. Han, D. Yu, and Q. Cao, "Assessment on the features of coupling interaction of the food-energy-water nexus in china," *J. Cleaner Prod.*, vol. 249, 2019, Art. no. 119379.
[36] W. Ziyu, F. Nando, de, and L. Marc, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Representations*, 2016.
[38] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, "Energy-efficient scheduling for real-time systems based on deep Q-learning model," *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 132–141, Jan. 2019.
[39] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.
[40] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, May 2017, Art. no. 84–90.
[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015.
[44] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, "Edge QoE: Computation offloading with deep reinforcement learning for internet of things," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9255–9265, Oct. 2020.
[45] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, Aug. 2019.

Min Xue received the bachelor's degree from the Qingdao University of Science and Technology, Qingdao, China, in 2019. She is currently working toward the master's degree at the Center for Applied Mathematics, Tianjin University, China. Her research interests include deep learning, deep reinforcement learning, cloud computing and mobile edge computing.

Huaming Wu (Member, IEEE) received the BE and MS degrees in electrical engineering from the Harbin Institute of Technology, China, in 2009 and 2011, respectively, and the PhD degree with the highest honor in computer science from Freie Universität Berlin, Germany, in 2015. He is currently an associate professor with the Center for Applied Mathematics, Tianjin University. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing, edge computing and complex networks.

Guang Peng received the BE and MS degrees in electrical engineering from Xi'an Jiaotong University, China, in 2014 and 2016, respectively. He is currently working toward the PhD degree at Freie Universität Berlin, Germany. His research interests include intelligent computation, multi-objective optimization, mobile cloud/edge computing and deep learning.

Katinka Wolter received the PhD degree from Technische Universität Berlin, in 1999. She has been assistant professor with Humboldt-University Berlin and lecturer with Newcastle University before joining Freie Universität Berlin as a professor for dependable systems in 2012. Her research interests include model-based evaluation and improvement of dependability, security and performance of distributed systems and networks. He is an associate member of the IEEE.