Future Generation Computer Systems 102 (2020) 847–861

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning

Haifeng Lu, Chunhua Gu *, Fei Luo, Weichao Ding, Xinping Liu

School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

HIGHLIGHTS

- This paper investigates the offloading problem of multiple service nodes for the cluster and multiple dependencies for mobile tasks in large-scale heterogeneous mobile edge computing.
- The problem of task offloading in mobile edge computing is solved by improved deep reinforcement learning based on LSTM and candidate networks.
- This paper analyzes the advantages and disadvantages of the offload strategy by comprehensively comparing energy consumption, cost, load balancing, latency, network usage and average execution time.

ARTICLE INFO

Article history:
Received 27 March 2019
Received in revised form 26 May 2019
Accepted 10 July 2019
Available online 12 July 2019

Keywords:
Mobile edge computing
Task offloading
Deep reinforcement learning
LSTM network
Candidate network

ABSTRACT

With the maturity of 5G technology and the popularity of intelligent terminal devices, the traditional cloud computing service model cannot deal with the explosive growth of business data quickly. Therefore, the purpose of mobile edge computing (MEC) is to effectively solve problems such as latency and network load. In this paper, deep reinforcement learning (DRL) is first proposed to solve the offloading problem of multiple service nodes for the cluster and multiple dependencies for mobile tasks in large-scale heterogeneous MEC. Then the paper uses the LSTM network layer and the candidate network set to improve the DQN algorithm in combination with the actual environment of the MEC. Finally, the task offloading problem is simulated by using iFogSim and Google Cluster Trace. The simulation results show that the offloading strategy based on the improved IDRQN algorithm has better performance in energy consumption, load balancing, latency and average execution time than other algorithms.

1. Introduction

With the rapid development of mobile communication technology and the popularization of smart devices, more and more smartphones, laptops and sensors can access remote services through the Internet. According to Cisco's statistical report, personal mobile Internet traffic will gradually increase at a compound annual growth rate of 47% from 2016 to 2021 [1]. It is estimated that global mobile data traffic will reach 46.6 Exabyte per month by the end of 2021, of which smartphones will account for 86% [2]. Each person will generate 1.7 MB data size per second in 2020 according to the research of ITU Telecommunication Standardization Sector [3]. In addition, due to the increasing popularity of 5G networks, the three major types of services, including enhanced mobile broadband, massive machine communication, and ultra-reliable low-latency communication, pose great challenges to operators' transmission networks and core networks. Because these application scenarios can bring users higher bandwidth rates, more network connections, and lower latency, which will result in an explosive growth of data size and business requests that the core network needs to process per unit time. Therefore, with the increasing network traffic carried by mobile devices and IoT devices, traditional service mode of the cloud data center cannot meet the daily needs of users. On the one hand, users need to interact with the data center when acquiring services, so the network latency caused by the relative distance between the user and the data center will have a great impact on some latency-sensitive applications, such as network games, virtual reality, and video communications. On the other hand, since the data interaction generated by all applications needs to go through the core network, it will put a lot of pressure on network load during the peak periods of service access [4]. Compared to the limitations of cloud computing in terms of network, MEC provides services close to physical entities or data sources by using an open platform that integrates

* Corresponding author.
E-mail address: chgu@ecust.edu.cn (C. Gu).

https://doi.org/10.1016/j.future.2019.07.019
0167-739X/

network, computing, storage and application core capabilities, so that its applications can be executed on the edge side, resulting in faster network service response and satisfying the industry's real-time processing, intelligent application, security, and privacy protection [5]. In other words, the core idea of MEC is to marginalize and localize computing resources and cache resources, so as to reduce network latency and alleviate bandwidth pressure [6]. This not only satisfies the need for mobile devices to expand computing capacity but also makes up for the shortcomings of long transmission latency in cloud computing.

Since different tasks have different calculation amount and data transmission amount, in order to ensure that mobile devices can improve performance through task offloading, it is necessary to use a task offloading strategy to perform the metric measurement, and determine whether the task is executed at the local mobile or at the remote server [7]. At the same time, because the computing, storage, bandwidth and other resources of the MEC server are limited, in order to reduce the latency in the network and make better use of the limited resources of the edge server cluster, the task offloading strategy also needs to determine its target server for offloading [8]. Currently, there are two main types of task offloading: coarse-grained task offloading and fine-grained task offloading [9]. Coarse-grained task offloading takes a mobile application as the offloading object and does not divide it into multiple subtasks according to its functions. This method often does not fully consider transmission latency and transmission consumption of the whole application. Fine-grained task offloading refers to first dividing a mobile application into multiple subtasks with data dependencies, because divided subtasks require less computational complexity and the amount of data transfer, some or all subtasks can be offloaded to multiple remote servers for processing, which saves computing time and transmission time, and has a higher resource utilization rate for the edge server cluster.

Therefore, this paper proposes a Markov Decision Process (MDP) model based on MEC environment, which uses deep reinforcement learning to solve the fine-grained task decision-making problem of what to offload, how much to offload and where to offload. Considering the scenarios of multi-service nodes and mobile subtasks of multiple dependencies in large-scale heterogeneous MEC, this solution proposes a fine-grained task offloading scheme based on deep reinforcement learning, which can reduce latency, cost, energy consumption and network usage of the MEC platform.

The rest of the paper is organized as follows. In Section 2, related works are presented. The problem model is described in Section 3. The proposed task offloading algorithm is presented in Section 4. In Section 5, the experiment results of the simulation are provided. Section 6 presents the conclusion of this paper.

2. Related work

In order to save the energy consumption of mobile devices, expand their computing power, effectively utilize the computing resources of edge servers and the cloud data center, many references have studied the offloading problem of MEC in recent years. Ref. [10] combines MEC with wireless charging technology, studies the offloading problem of wireless charging IoT devices, and proposes an online scheduling strategy based on Lyapunov optimization. Ref. [11] solves the task offloading problem by using mixed integer nonlinear programming, which ultimately reduces the user cost. Ref. [12] combines the medical network physical system with fog calculation to comprehensively optimize costs from multiple aspects such as base station association, task assignment, and virtual machine placement. Ref. [13] first proposes the concept of fog population based on hierarchical optimization process and then combines it with a genetic algorithm to determine whether the service is placed in the current population for execution or propagated to the neighboring population for execution. Ref. [14] presents a model for predicting the number of user requests in edge computing and designs a task offloading scheme based on this model. The effectiveness of this scheme is verified by comparing the performance of the greedy algorithm, linear programming and genetic algorithm in task offloading. Ref. [15] establishes a mathematical model of MEC architecture., the MEC offloading strategy is optimized by measuring the round trip time between user's device and MEC server at the edge of the network, which determines when to offload user's computational tasks to MEC server for processing and verifies the effectiveness of the policy through the face recognition application. Compared with the local execution of mobile devices, the service latency is greatly reduced and the energy consumption of devices is saved. Ref. [16] studies the multi-user service latency problem in MEC offloading scenario and proposes a new partial computing offloading model. The optimal strategy is used to optimize the allocation of communication and computing resources. Experiments are carried out in specific scenarios where communication resources are much larger than computing resources. Compared with the local execution and the edge execution of devices, the proposed partial offloading strategy can minimize the latency of all user devices, thereby improving the user quality of service. Ref. [17] considers a scenario in which a single user offloads computational tasks to multiple edge nodes, and optimizes edge node selection, offloading order, and task allocation through heuristic algorithms, reconstruction linearization techniques, and semi-deterministic relaxation algorithms to achieve the balance between latency performance and reliability performance in MEC. In Ref. [18], the multi-user MEC architecture in high-reliability and low-latency scenarios is studied. The user's task queue is analyzed by introducing probability and extreme value theory, and Lyapunov stochastic optimization is used to the minimization problem of calculation and transmission energy.

According to the above research, heuristic algorithm and meta-heuristic algorithm are widely used to solve NP-Hard problems such as task offloading, but both of them have their own shortcomings. Among them, the heuristic algorithm is easy to fall into a local minimum, and it is difficult to get the overall optimal results; the meta-heuristic algorithm has too many parameters, the calculation results are difficult to reusable, and the parameter tuning cannot be performed quickly and effectively. In contrast, deep reinforcement learning has characteristics of self-learning and self-adaptation by combining the advantages of deep learning and reinforcement learning. It needs to provide fewer parameters and has better global search capabilities, which can solve more complex, high-dimensional and more realistic task scenarios. However, the results of deep reinforcement learning algorithm must depend on the complete state information, and the overestimation of its value caused by training errors will also affect the performance of the algorithm. Therefore, this paper improves the DQN algorithm by using the LSTM network and the candidate network to solve the defect of deep reinforcement learning, so as to solve the problem of MEC task offloading with a large number of mobile devices.

3. Problem model

3.1. Task dependency model

The structure of the MEC usually includes three parts: the cloud data center layer, the edge server layer, and the mobile device layer. As shown in Fig. 1, the mobile device layer includes sensors, laptops, mobile phones and other devices with

Table 1
Major notations.

Symbol	Definition
D	Cloud data center
E	Edge server set
G	Mobile device set
m	Number of edge server
n	Number of mobile device
C_i^{in}	Input data size of the ith subtask
N_i^{do}	Downlink bandwidth of the ith subtask
M_i	CPU resource required for the ith subtask deployment
C_i^{out}	Output data size of the ith subtask
N_i^{up}	Uplink bandwidth of the ith subtask
SL_i^{tran}	Data transmission latency of the ith subtask
SL_i^{comp}	Calculation latency of the ith subtask
$appnum$	Number of all mobile applications
$subnum_j$	Number of subtasks obtained by dividing the jth application
U_i	CPU utilization of the ith computing device
S_t	State space at time step t
A	Action space
R	Reward function
T	State transition function
Z	Observation set
O	Observation function
$\hat{Q}$	Approximate reward function
n'	Size of candidate network set
m'	Size of Net_1
$L(\theta)$	Loss function
MainNet	Current neural network
TargetNet	Target neural network

low processing performance; The edge server layer divides all edge servers into regions according to their relative distances, each region contains some heterogeneous edge servers of moderate performance; The cloud data center layer contains a large number of high-performance physical servers that form a cluster. When the mobile device needs to improve performance through task offloading, a whole mobile application will be first divided into several subtasks by some segmentation algorithm. Some of these subtasks must be executed locally, such as user interaction tasks, device I/O tasks, and peripheral interface tasks. There are also some tasks that can be offloaded to servers for execution, they are usually data-processing tasks with a large amount of computation. The divided subtasks have data interaction with each other and can be executed independently, which is a prerequisite for fine-grained offloading decisions.

If there is a dependency between subtasks after the mobile application is divided, the relationship can be represented by a directed graph Loop = (S, B). Each node $s_i \in S$ of the graph represents the divided subtask, and each edge $b_{uv} \in B$ of the graph represents the transition data between tasks. For example, b_{ij} indicates that the task s_j must receive the processing result of task s_i before it can continue to execute. As shown in Fig. 2, the set of subtasks after dividing a mobile application is $S = \{s_1, s_2, s_3, s_4, s_5\}$, where s_1 and s_5 must be executed on the local device, and remaining subtasks can be offloaded as needed. At the same time, directed arrows are used to indicate data dependence among subtasks. For example, the s_5 must receive the data of s_2 and s_4 before it can continue to execute. In addition, the major notations used in this paper are summarized in Table 1.

3.2. Resource consumption model

In this paper, all computing devices in the MEC are represented by (D, E, G), where D represents a cloud data center with massive computing resources, which is mainly composed of a physical machine cluster; $E = \{e_1, e_2, \ldots, e_m\}$ represents a set of edge servers whose number is m; $G = \{g_1, g_2, \ldots, g_n\}$ represents a set of mobile devices whose number is n, and it is set that only one application request can be sent by each device in a certain period of time. The divided ith subtask is represented by a five-tuple $\text{Sub}_i = \{C_i^{in}, N_i^{do}, M_i, C_i^{out}, N_i^{up}\}$, where C_i^{in} represents the amount of input data transferred from the previous subtask to the current subtask; N_i^{do} represents the downlink bandwidth that the current subtask can use to receive data; M_i represents the computing resources required for this subtask deployment, because one of the optimization objectives of this paper is the total energy consumption generated by the task processing, and some studies show that the correlation coefficient between the power consumption of the host and its CPU utilization is as high as 0.990667, so this value is defined as the CPU processing performance required for task deployment [19]. C_i^{out} represents the amount of result data transferred from the current subtask to the next subtask; N_i^{up} represents the uplink bandwidth used by the current subtask for uploading processing results. In view of the offloading decision generated by the above task model, this paper designs the following model to evaluate its effectiveness in many aspects:

3.2.1. Energy consumption model

For the total energy consumption of all computing devices including smartphones, sensors and remote servers in a certain execution time, it mainly consists of two parts: computing energy consumption $Energy^{comp}$ of all devices and offloading energy consumption $Energy^{off}$ of mobile devices. This paper first defines the computing energy consumption model of the ith computing device as follows [19]:

$$P_i(u) = \begin{cases} K * P_i^{max} + (1-K) * P_i^{max} * u & if\ u > 0 \\ 0 & otherwise \end{cases} \tag{1}$$

where K represents the percentage of the idle state of the computing device to the full load state, P_i^{max} represents the energy consumed by the ith computing device at full load, and u is the CPU utilization. In addition, the load of the computing device is constantly changing with time. Now suppose that u(t) is a function for calculating the CPU utilization of the device per unit time. From time t_0, the computing energy consumption of a computing device in unit time t is:

$$\begin{cases} Energy_i^{comp}(t_0) = \int_{t_0}^{t_0+t} P_i(\text{u(t)})dt \\ \text{u(t)} = \min\{1, \dfrac{\sum_{j=1}^{alltask_i} M_j}{DR_i}\} \end{cases} \tag{2}$$

where $alltask_i$ represents the number of all tasks in the ith computing device; M_j represents the CPU resource required for jth subtask deployment; DR_i represents the total CPU resource of the ith device. At the same time, the data transmission rate r_j of the jth mobile device in a certain channel is defined as [20]:

$$r_j = W \log_2(1 + \frac{p_j h_j^2}{N_0}) \tag{3}$$

where W represents the fixed transmission bandwidth of the channel; N_0 represents the received average power of interference plus additive background Gaussian noise; p_j represents the energy consumption of the jth mobile device for transmitting data; h_j represents the channel gain of the jth mobile device,

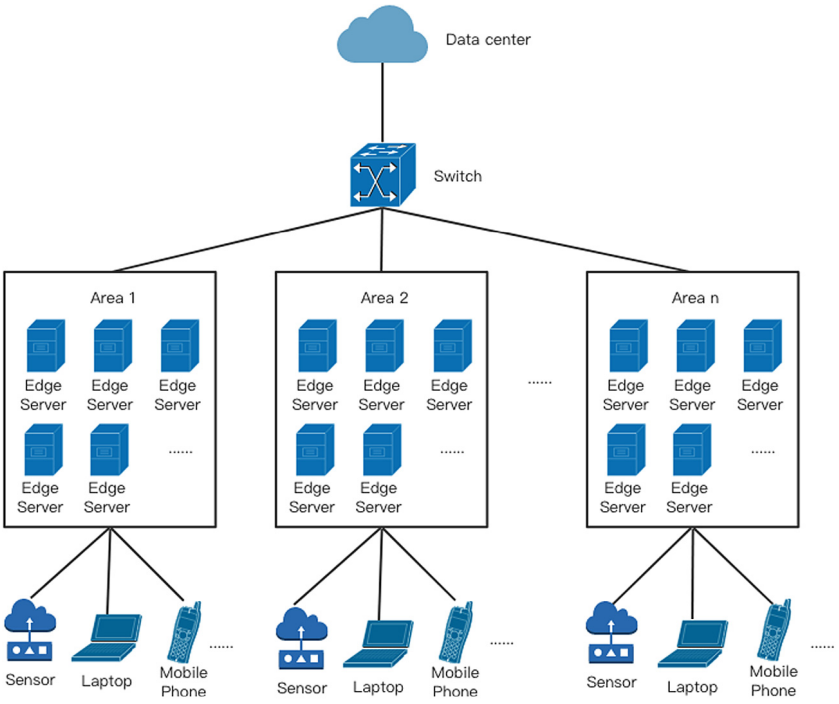


Fig. 1. Structural diagram of MEC.

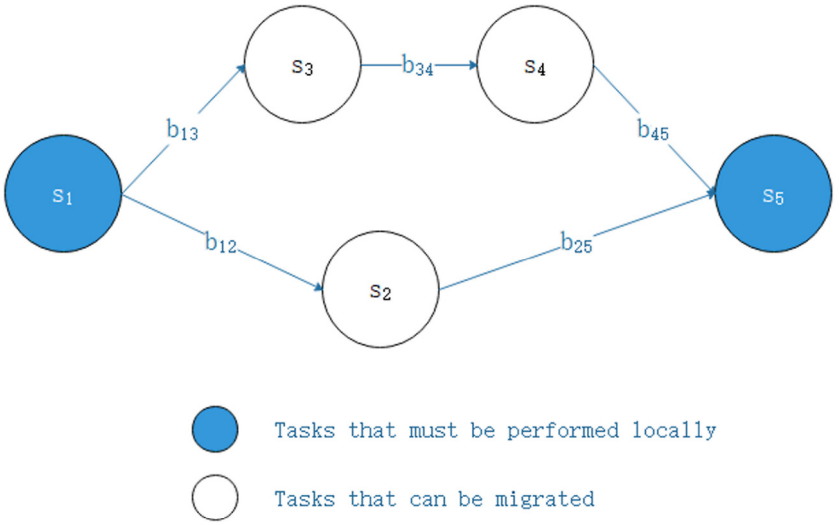


Fig. 2. Data dependency of subtasks.

which is a fixed value during the offloading process. Since setting the data transmission rate of mobile devices as a constant is the most energy-saving transmission strategy under time constraints, the data transmission rate r_j of the jth mobile device in unit time t is:

$$r_j(t_0) = \frac{\int_{t_0}^{t_0+t} \sum_{k=1}^{offnum_j} C_k^{out}(t)dt}{t} \tag{4}$$

where $offnum_j$ represents the number of subtasks that the jth mobile device needs to be offloaded; $C_k^{out}(t)$ represents the amount of data that the kth subtask needs to upload. At the same time, according to formula (3), $f(x)$ is defined as the energy consumption p_j of data transmission in the jth mobile device, and x is the data transmission rate $r_j(t_0)$ in the unit time, then the offloading energy consumption of the jth mobile device in unit time t is as follows [20]:

$$\begin{cases} (x) = \dfrac{N_0(2^{\frac{x}{W}} - 1)}{h_j^2} \\ Energy_j^{off}(t_0) = p_j t = f\left(r_j(t_0)\right) t = \dfrac{N_0 t(2^{\frac{r_j(t_0)}{W}} - 1)}{h_j^2} \end{cases} \tag{5}$$

All computing devices with a total number of (1+m+n) and mobile devices with a total number of n generate computing energy consumption and offloading energy consumption in unit time t as follows:

$$Energy_{sum}(t_0) = \sum_{i=1}^{1+m+n} Energy_i^{comp}(t_0) + \sum_{j=1}^{n} Energy_j^{off}(t_0) \tag{6}$$

3.2.2. Cost model

The user needs to pay the corresponding fee for the computing resources provided by the remote server. This paper uses the dynamic price model based on the remaining amount of resources. The lower the remaining amount of resources, the higher the resource price. At this time, the user prefers to select the service node with the lower unit price as the offloading target, so as to reduce user costs and improve resource utilization. Formula (7) is a dynamic price model based on the remaining amount of resources in unit time t [21]:

$$Cost_i\,(t_0) = \mathrm{CC} + \mathrm{UT}^*\mathrm{RPM}^*\mathrm{TM}^* \int_{t_0}^{t_0+t} \mathrm{LU(t)}dt \tag{7}$$

CC represents the cost that the current device has generated; UT represents the interval time for the fee calculation, here is the unit time t; RPM represents the unit price of computing resources; TM represents the total computing resource of the current device; LU(t) represents the computing resource ratio that the current device has used per unit time. At the same time, since computing resources of the local device belong to users themselves and do not need to calculate the cost, so the total cost of all remote devices (the quantity is 1+m) is as follows:

$$Cost_{sum}(t_0) = \sum_{i=1}^{1+m} Cost_i(t_0) \tag{8}$$

3.2.3. Load balancing

Load balancing is an important method to realize the effective use of various resources in a cluster. It is mainly for the purpose of optimizing resource usage, maximizing throughput, minimizing response time, avoiding overload, enhancing network data processing capability, and improving network flexibility and availability. Formula (9) is used to calculate the load balancing of all devices in the cluster [22]:

$$\begin{cases} Load_i(t_0) = \int_{t_0}^{t_0+t} (\sum_{k=1}^{lb} U_k * L_k(t))dt \\ avg(t_0) = \frac{\sum_{i=1}^{1+m+n} Load_i(t_0)}{1+m+n} \\ LB(t_0) = \sqrt{\frac{\sum_{i=1}^{1+m+n} (Load_i(t_0) - avg(t_0))^2}{1+m+n}} \end{cases} \tag{9}$$

$Load_i(t_0)$ represents the synthetical load of all resources in the ith computing device during t_0; lb represents the number of indicators used to calculate the load balancing, such as CPU, memory, and hard disk, and this paper mainly considers the CPU utilization of computing devices; U_k is the weight of each resource, which satisfies the condition $\sum_{k=1}^{lb} U_k = 1$; $L_k(t)$ represents the usage rate of each resource per unit time; $avg(t_0)$ is the average load of all computing devices; $LB(t_0)$ represents the load value of all computing devices, the smaller the value, the better the load balancing result.

3.2.4. Average service latency

According to the dependency between subtasks, the average service latency SL^{avg} of all mobile application consists of two parts: (1) the data transmission latency SL^{tran} between subtasks; (2) the data computation latency SL^{comp} generated by processing business data of subtasks. Since the amount of data transmission and available bandwidth of the subtask change dynamically with time. Therefore, starting from the time t_0, the transmission latency of the ith subtask for downloading dependent data in unit time t is:

$$SL_i^{do}(t_0) = \int_{t_0}^{t_0+t} \frac{C_i^{in}(t)}{N_i^{do}(t)}dt \tag{10}$$

The transmission latency of the ith subtask for uploading results in unit time t is:

$$SL_i^{up}(t_0) = \int_{t_0}^{t_0+t} \frac{C_i^{out}(t)}{N_i^{up}(t)}dt \tag{11}$$

Then the data transmission latency of the ith subtask in unit time t is [23]:

$$\begin{cases} SL_i^{tran}(t_0) = x_{tran}^{do} \int_{t_0}^{t_0+t} \frac{C_i^{in}(t)}{N_i^{do}(t)}dt + x_{tran}^{up} \int_{t_0}^{t_0+t} \frac{C_i^{out}(t)}{N_i^{up}(t)}dt \\ x_{tran}^{do} = 1\{l_i \neq l_{i-1}\} \\ x_{tran}^{up} = 1\{l_i \neq l_{i+1}\} \end{cases} \tag{12}$$

where l_{i-1}, l_i, and l_{i+1} represent the devices for deploying the previous subtask, the current subtask, and the next subtask, respectively. x_{tran}^{do} represents whether the current subtask needs to download dependent data through the network, and x_{tran}^{up} represents whether the current subtask needs to upload the processing result to the network. $1\{\cdot\}$ is Iverson bracket, which is equivalent to 1 when the condition is satisfied. Otherwise, it is equivalent to 0. In addition, the calculation latency of the ith subtask generated by processing data in unit time t is:

$$\begin{cases} SL_i^{comp}(t_0) = x_{off} \int_{t_0}^{t_0+t} \frac{C_i^{in}(t)}{f^{server}(t)}dt + (1 - x_{off}) \int_{t_0}^{t_0+t} \frac{C_i^{in}(t)}{f^{local}(t)}dt \\ x_{off} = 1\{l_i \neq l_{local}\} \end{cases} \tag{13}$$

where $f^{server}(t)$ represents the processing rate of the remote server for deploying the ith subtask; $f^{local}(t)$ represents the processing rate of the local device for deploying the ith subtask; x_{off} represents whether the current subtask is offloaded to the remote server for execution. Based on the above analysis, the average latency of all mobile applications in unit time t is:

$$\begin{aligned} SL^{avg}\,(t_0) &= \frac{SL^{total}\,(t_0)}{appnum} \\ &= \frac{\sum_{j=1}^{appnum} \sum_{i=1}^{subnum_j} \left(SL_i^{tran}\,(t_0) + SL_i^{comp}\,(t_0)\right)}{appnum} \end{aligned} \tag{14}$$

where $SL^{total}\,(t_0)$ represents the total latency of all mobile applications; $appnum$ represents the number of mobile applications; $subnum_j$ represents the number of subtasks after the jth mobile application is divided.

3.2.5. Average execution time

Execution time refers to the time it takes for a subtask to process data. The more resources subtasks can use, the shorter the processing time, the more requests can be processed in unit time. Therefore, the offloading decision needs to reduce the average execution time of all mobile application as much as possible. The formula for the average execution time is:

$$\mathrm{ET}\,(t_0) = \frac{\sum_{j=1}^{appnum} \sum_{i=1}^{subnum_j} SL_i^{comp}(t_0)}{appnum} \tag{15}$$

where $\mathrm{ET}\,(t_0)$ represents the average execution time of all mobile applications per unit time.

3.2.6. Network usage

The network usage refers to the amount of data transmission generated by all mobile applications in unit time t. If the network usage is too high, the entire network may be congested, further reducing the processing performance of the computing device.

The calculation formula for network usage per unit time is as follows [23]:

$$\begin{cases} TA(t_0) = \sum_{j=1}^{appnum} \sum_{i=1}^{subnum_j} \int_{t_0}^{t_0+t} \left(x_{tran}^{do} C_i^{in}(t) + x_{tran}^{up} C_i^{out}(t) \right) dt \\ x_{tran}^{do} = 1\{l_i \neq l_{i-1}\} And\ x_{tran}^{up} = 1\{l_i \neq l_{i+1}\} \\ \mathrm{NU}(t_0) = \dfrac{SL^{total}(t_0) * TA(t_0)}{t} \end{cases} \tag{16}$$

where $TA(t_0)$ represents the total amount of data transmission generated by all mobile application per unit time; $\mathrm{NU}(t_0)$ represents the network usage per unit time.

4. Algorithmic design

Reinforcement learning is an algorithmic model that can make optimal decisions through self-learning in a specific scenario, it models by abstracting all real-world problems into an interactive process between the agent and the environment. At each time step in the interaction process, the agent receives the state of the environment and selects the corresponding response action. Then in the next time step, the agent obtains a reward value and a new state according to the feedback of the environment. Reinforcement learning is more adaptable to the environment based on the rewards of continuous learning. The goal of all agents is to maximize the sum of the expected rewards obtained at all time steps [24]. While reinforcement learning has many advantages, it is also lacking in scalability and is inherently limited to fairly low-dimensional problems. These limitations exist because the reinforcement learning algorithm has the same memory complexity, computational complexity, and sample complexity as other algorithms. In order to solve the difficult decision-making problem in reinforcement learning, deep reinforcement learning combines the perception ability of deep learning with the decision-making ability of reinforcement learning. It relies on powerful function approximation and the expressive learning property of deep neural network to solve the environmental problems with high-dimensional state space and action space [25]. In the following, the MDP model is constructed by combining the actual MEC environment, and the DQN algorithm is improved to solve the task offloading problem in the MEC.

4.1. MDP model

4.1.1. State space

In order to comprehensively consider the characteristics between subtasks and server resources in MEC, this paper defines the state space at time step t as $S_t = (C^{in}, N^{do}, \mathrm{M}, C^{out}, N^{up}, U_1, U_2, \ldots, U_i, \ldots, U_{2+m})$, Where C^{in} represents the amount of input data required for the current subtask; N^{do} represents the downlink bandwidth that the subtask can use to download the result; M represents the CPU resources required for the deployment of the subtask, C^{out} represents the amount of result data generated by the current subtask, and N^{up} represents the uplink bandwidth that the subtask can use to upload the data; U_i represents the CPU utilization of the ith computing device at time step t. In order to ensure that the subtask can only be selected to be executed on the local mobile device or the remote server, the offloading decision of the subtask only needs to consider computing devices with the number of (2+m), which includes one cloud data center, one local mobile device, and m edge servers.

4.1.2. Action space

In order to offload the subtask to the appropriate computing device, the action space is specified in a one-to-one correspondence with the set of computing devices in the deep reinforcement learning, and $(0/1)_i^j$ is used to indicate whether the ith subtask is offloaded to the jth computing device. For example, the action space $A_i = (0, 0, 1, \ldots, 0)$ indicates that the ith subtask is offloaded on the third computing device according to the policy. Therefore, the action space is $A = (a_1, a_2, \ldots, a_{2+m})$ for a cluster containing (2+m) computing devices. In addition, the preprocessing of the action space ensures that the deep reinforcement learning algorithm can learn valid actions in the iterative process, so as to avoid the reward function need to set penalty values for invalid actions, reduce the complexity and calculation of the reward function, and accelerate the learning rate of the optimal strategy. Comparing the resources required for the subtask to be offloaded with the available resources of the target computing device. If the computing device can satisfy the offloading requirements, set its corresponding action space to True, otherwise set to False. Algorithm 1 is pseudo code for an action space that can perform valid offloading operations.

4.1.3. Reward function

This paper will consider the three aspects of energy consumption, cost and load balancing of all devices in MEC to evaluate the advantages and disadvantages of the offloading decision. At the same time, in order to solve the deviation of reward value caused by the performance difference of heterogeneous computing devices in each time step, this paper uses z-score standardization method to normalize energy consumption, cost, and load balancing respectively. When the sequence is $x_1, x_2, \ldots, x_{num}$, the formula is as follows [26]:

$$\begin{cases} \bar{x} = \dfrac{1}{num} \sum_{i=1}^{num} x_i \\ s = \sqrt{\dfrac{1}{num-1} \sum_{i=1}^{num} (x_i - \bar{x})^2} \\ Z_i = \dfrac{x_i - \bar{x}}{s} \end{cases} \tag{17}$$

By combining Formulas (6), (8), and (9) with Formula (14), respectively, the normalization value of energy consumption, cost, and load balancing of all computing device can be obtained by comparing with their respective historical data sets. In addition, this paper focus on minimizing the total resource consumption over time, so the reward function during t_0 is defined as:

$$\begin{cases} \mathrm{R} = -(\alpha Z_{eg}(t_0) + \beta Z_{cs}(t_0) + \gamma Z_{lb}(t_0)) \\ \alpha + \beta + \gamma = 1 \end{cases} \tag{18}$$

where $Z_{eg}(t_0)$, $Z_{cs}(t_0)$, and $Z_{lb}(t_0)$ represent the normalization values of energy consumption, cost, and load balancing of all devices during the time t_0, respectively; α, β, and γ represent the weights of the three factors, and their sum is 1. In addition, according to the MDP model constructed above, the pseudo code of the MEC task offloading strategy combined with deep reinforcement learning is as follows:

4.2. Algorithm optimization

4.2.1. Optimization based on LSTM network

DQN is a deep reinforcement learning algorithm based on value iteration whose goal is to estimate the Q value of the optimal strategy. The algorithm calculates approximation function by using the deep neural network, and transforms Q-Table updating problem into a function fitting problem, so that it can get similar

Algorithm 1 Valid action space algorithm

Input: The subtask T that need to be offloaded, all physical machines hostList that include the local mobile device, edge servers and a cloud data center

Output: Valid action space

1. Initialize the valid action space A_{valid} based on hostList
2. Size = size of the valid action space A_{valid}
3. $\text{Resource}_{request}$ = Get the resources needed for the current subtask T deployment
4. for i=0, Size-1 do
5. $\text{Resource}_{available}$= Get the available resources of the current device hostList[i]
6. if $\text{Resource}_{request} > \text{Resource}_{available}$ then
7. $A_{valid}[\text{i}] = \text{False}$
8. else
9. $A_{valid}[\text{i}] = \text{True}$
10. end if
11. end for
12. return A_{valid}

output actions according to the current state, so as to solve the shortcomings of traditional Q-Learning algorithm in high-dimensional and continuous problems. As shown in the following formula, the calculation result of the $\hat{Q}$ function is approximated to the Q value by updating the parameter θ [24]:

$$\begin{cases} \hat{Q}\left(s_t, a_t, \theta\right) \approx Q\left(s_t, a_t\right) \\ Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha(r_{t+1} + \gamma max_{a'} Q\left(s_{t+1}, a'\right) - Q\left(s_t, a_t\right)) \end{cases} \tag{19}$$

where s_{t+1} represents the next state after the action a_t is taken at time step t, r_{t+1} is the immediate reward after taking action a_t, and a' is all actions that can be taken for state s_{t+1}; γ is the discount coefficient in the process of value accumulation; α is the learning rate, and the larger the value, the smaller the impact of historical training results.

DQN not only improves the search speed of Q-Learning algorithm by function fitting, but also improves its diversity and stability by adding experience pool and the target network. The experience pool stores the transfer samples (s_t, a_t, r_t, s_{t+1}) obtained by the interaction between the agent and the environment at each time step into the memory space. When training, a certain number of samples are randomly selected to solve the problem of data correlation and non-static distribution; The target network value *TargetQ* refers to the target Q value of the training process generated by using another network TargetNet. The structure of the TargetNet is consistent with the neural network MainNet of the DQN, and the parameters of MainNet are copied to TargetNet after C round iteration. Therefore, the current Q value and the target Q value are used to calculate the loss function by maintaining the difference of the two network parameters for a period of time, and then the parameters of the network MainNet are inversely updated using a method such as stochastic gradient descent (SGD). The loss function of the DQN algorithm is calculated as:

$$\begin{cases} L\left(\theta\right) = \text{E}[(\text{TargetQ} - \hat{Q}\left(s_t, a_t, \theta\right))^2] \\ TargetQ = r_{t+1} + \gamma max_{a'} \hat{Q}\left(s_{t+1}, a', \theta^-\right) \end{cases} \tag{20}$$

where $\hat{Q}\left(s_t, a_t, \theta\right)$ is the output of the current network MainNet, which is used to calculate the Q value of the current state–action pair. $\hat{Q}\left(s_{t+1}, a', \theta^-\right)$ represents the output of the target network TargetNet, which is used to calculate target Q values after taking all optional actions.

In the real environment of MEC, it is difficult for the system to directly obtain the accurate state in the current time step due to the complexity of the problem and the limitations of perception. Assuming that the state information of the system cannot be directly observed, it is partially known. Therefore, it is usually necessary to use a POMDP (Partial Observation Markov Decision Process) to model a system and make decisions with only incomplete state information. POMDP can be described by a six-tuple (S, A, T, R, Z, O), where S represents the state set of the system's environment and is partially observable; A represents a finite set of actions; Z represents a finite set of observations; $\text{T}: \text{S} \times \text{A} \rightarrow \Pi(\text{S})$ is the state transition function; $\text{R}: \text{S} \times \text{A} \rightarrow \text{R}$ is a reward function; $\text{O}: \text{S} \times \text{A} \rightarrow \Pi(\text{Z})$ is the observation function given by state and action. Under normal circumstances, DQN can only achieve good results when the observation $z \in Z$ can reflect the real environment state $s \in S$ well, so it is difficult to directly solve the actual MEC problem.

In view of the gradual change of resources over time in MEC and the memory ability of LSTM network for long-term state, this paper proposes to combine LSTM and DQN to deal with the time-dependent task offloading problem. The recurrent structure

Algorithm 2 Task Offloading Algorithm Combined with Deep Reinforcement Learning

Input: Requested all physical machines hostList that include the local mobile device, edge servers and a cloud data center, all subtasks taskList that need to be offloaded

Output: The offloading decision for taskList

1. Initialize offloading list offloadList for taskList
2. Initialize the action space A based on hostList
3. for subtask T that T ∈ taskList do
4. Get input data size C^{in}, download bandwidth N^{do}, required CPU resources M, output data size C^{out}, upload bandwidth N^{up} of the subtask T
5. Get cpu utilization CPUList of hostList in time step t
6. Construct the state space S_t of the subtask T in time step t
7. A = Calculate the valid action space based on Algorithm 1
8. Select an optimal action $a_t \in A$ based on state S_t using a deep reinforcement learning algorithm, a_t maps to machine h ∈ hostList
9. Get mapping <T, h>
10. Remove subtask T from all subtasks taskList
11. Add mapping <T, h> to offloading list offloadList
12. end for
13. Return offloadList

is used to integrate any long-term historical data to estimate the current state more accurately by replacing the last fully connected layer of the DQN network with the LSTM layer. As shown in Fig. 3, the improved DRQN algorithm composes a state–action pair by the observation state z_t in the current time step and the action a_{t-1} in the previous time step, and integrates it with the output value in the LSTM to derive the real environment state s_t, then it is imported into the deep neural network for training. Therefore, compared to the $\hat{Q}(s_t, a_t, \theta)$ used by the DQN algorithm, DRQN prefers to use $\hat{Q}(z_t, h_t, \theta)$ for function fitting, where h_t represents the output value of the LSTM layer at the current time step. Its iteration formula is:

$$h_{t+1} = LSTM(h_t, z_t, a_{t-1}) \tag{21}$$

4.2.2. Optimization based on candidate network

The DQN algorithm guarantees the parameter difference between the current network and the target network by delayed updating, so as to increase the stability in the training process. However, formula (20) shows that DQN algorithm uses the same network in action evaluation and action selection. Therefore, when the value of an action in training process is overestimated because of noise or error, the value of the corresponding action will inevitably be overestimated when it is used for parameter updating in subsequent estimation. This overestimation of the value function will affect the stability of the algorithm, resulting in the generated strategy is not optimal. In order to overcome the problem of overestimation, this paper comprehensively considers results of multiple candidate networks to decouple the action selection and action value evaluation to ensure the optimal learning strategy [27].

As shown in Fig. 4, it is assumed that the candidate network set Net = $(net_1, net_2, \ldots, net_i, \ldots, net_{n'})$ can store networks with a total number of n', which consists of network sets Net_1 and Net_2. There are m' networks in the network set Net_1 and they are updated after satisfying the fixed iteration number C; The network set Net_2 has $(n' - m')$ networks that are selected for updating by comparing reward values. When the number of the network set Net_2 is less than $(n' - m')$, the current network generated by each iteration will be added to the Net_2 as a candidate network; When the number of the network set Net_2 is equal to $(n' - m')$, the current network and all networks in the network set Net_2 will train the currently selected state–action pair. If the reward value of the current network is greater than the minimum value of training results in the network set Net_2, the candidate network with the smallest reward value is replaced with the current network, otherwise the training is continued.

Because the training process of the DQN algorithm uses random sampling to train the neural network, different samples will form different target networks, and each target network has its

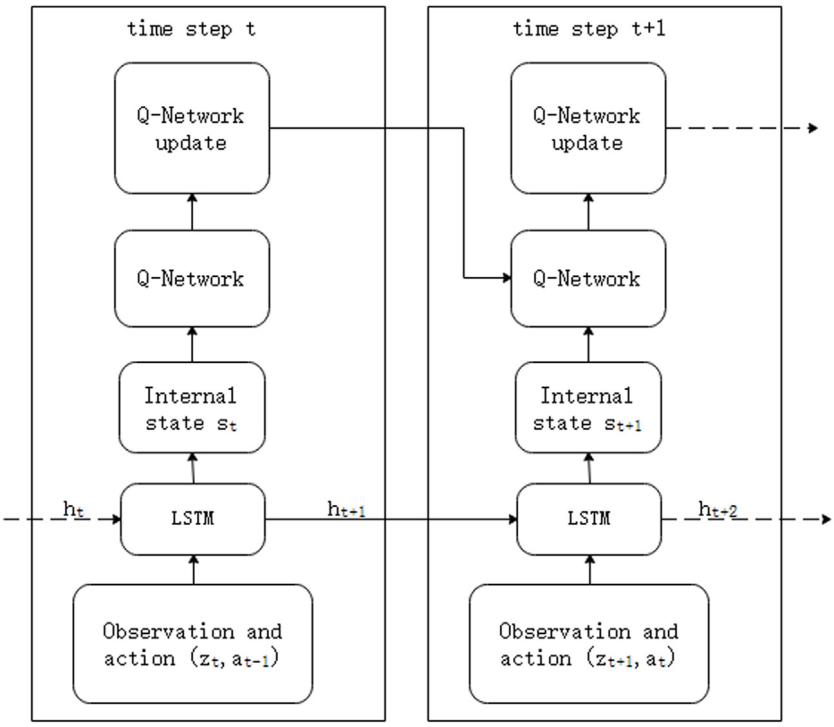


Fig. 3. DRQN algorithm training flowchart.

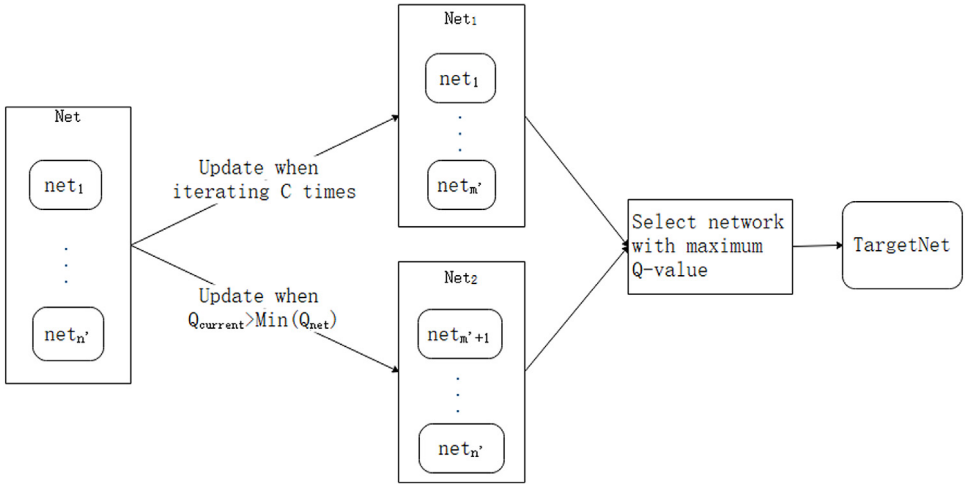


Fig. 4. Candidate network update flowchart.

Table 2
Computing device detailed configuration table.

Model	Type	CPU frequency (MHZ)	Core	Memory (GB)	Performance-to-power ratio (ssj_ops/watt)	Unit price
RX4770 M4	Cloud server	2100	112	192	12 828	0.05
RX350 S7	Edge server	2200	16	24	5 035	0.01
DL325 Gen10	Edge server	2000	32	128	8 083	0.01
DL360 Gen10	Edge server	2500	28	48	11 550	0.01
TX120	Mobile device	2666	2	4	454	0
TX150 S5	Mobile device	2666	2	4	356	0
TX150 S6	Mobile device	2400	4	8	667	0

advantage. In order to make full use of the state advantages of each target network based on different samples and iterations, this paper divides the candidate network set into a network set Net_1 which is updated according to the number of iterations and a network set Net_2 which is updated according to the reward value. The update frequency of the network in the set Net_1 is consistent with the update frequency of the target network, which helps to comprehensively consider the historical parameters of the target network; The network set Net_2 updates network parameters in real time with continuous iteration, replaces the network with the minimum reward value so as to keep the result optimal all the time. Finally, the reward values of all networks in the network

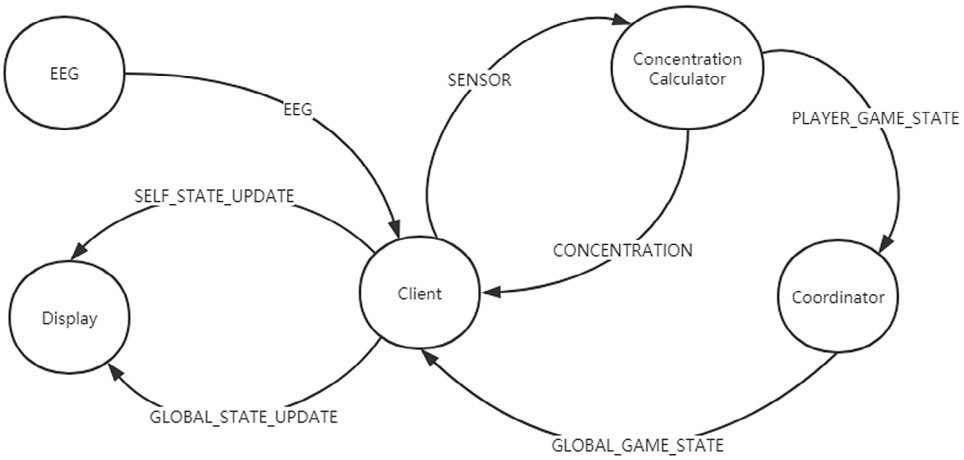


Fig. 5. VR game subtask dependency.

set Net are calculated separately, and network parameters with the maximum reward value of the current state–action pair are selected as parameters of the target network. Compared with formula (20), the calculation formula of the loss function is [28]:

$$\begin{cases} L(\theta) = \mathrm{E}[(\mathrm{TargetQ} - \hat{Q}(s_t, a_t, \theta))^2] \\ TargetQ = r_{t+1} + \gamma max_{a'}\{\max \hat{Q}_{net_i}(s_{t+1}, a', \theta_i)\} \end{cases} \tag{22}$$

The pseudo-code of the improved IDQN algorithm combined with candidate networks in each sampling process is as follows:

Table 3
Subtask parameter table.

Tuple type	CPU length	N/W length
EEG	2000	500
SENSOR	3500	500
PLAYER_GAME_STATE	1000	1000
CONCENTRATION	14	500
GLOBAL_GAME_STATE	1000	1000
GLOBAL_STATE_UPDATE	1000	500
SELF_STATE_UPDATE	1000	500

5. Simulation experiment

5.1. Simulation environment

In this paper, iFogSim is used to simulate the task offloading problem based on MEC, the advantages and disadvantages of the offloading decision are reflected by comparing the energy consumption, cost, load balancing, service latency, average execution time and network usage of each algorithm in a large-scale heterogeneous cluster. Implemented algorithms include the offloading strategy Mobile based on local device priority, the offloading strategy Edge based on edge server priority, the strategy DQN based on deep reinforcement learning, the improved deep reinforcement learning strategy HERDQN based on hindsight experience replay [28], the improved deep reinforcement learning strategy DRQN based on LSTM, the improved deep reinforcement learning strategy IDQN based on candidate network, the improved deep reinforcement learning strategy IDRQN based on candidate network and LSTM. The cluster of devices in the simulation experiment mainly consists of a cloud data center, 60 edge servers, and many mobile devices, in which all edge servers are divided into 10 different regions on average, and each mobile device can only send one offloading request in unit time. In this paper, the configuration and PPR(performance-to-power ratio) of all devices are set according to SPEC (Standard Performance Evaluation Corporation). In addition, the larger the PPR is, the less energy the device consumes at the same performance. The detailed information is shown in Table 2.

In order to simulate the offloading process after the mobile application is divided into different subtasks, this paper constructs subtask dependencies of a virtual reality game according to the Ref. [29]. As shown in Fig. 5, the application is mainly composed of five subtasks such as EEG, Client, Concentration Calculator, Coordinator and Display. The EEG and Display must be executed on the local device, and remaining subtasks can be offloaded according to the strategy. Latency-sensitive applications have very high requirements for offloading decisions. On the one hand, it is necessary to offload high-calculation modules to remote servers for execution to minimize the energy consumption of mobile devices; On the other hand, the computing modules need to be as close as possible to the data sources, so as to reduce the latency caused by data transmission between modules. As shown in Table 3, this paper uses the CPU Length and N/W Length to indicate the requirements of the task dependency on computational complexity and data throughput. When the CPU Length is larger and the N/W Length is smaller, it indicates that the task is more inclined to high computational demand. Otherwise, the task is more inclined to data transmission demand. At the same time, the parameters of all deep reinforcement learning algorithm are set uniformly to ensure the fairness of training results. Among them, the memory space of deep reinforcement learning is defined as M = 100 000, the learning rate of optimization algorithm SGD(Stochastic gradient descent) is $\alpha = 0.005$, the batch learning size K = 32, the updating period C = 50, and the discount coefficient $\gamma = 0.9$; For the improved HERDQN algorithm, the target utilization ratio of edge servers with DL360 Gen10 is set to 100%, because the PPR of the device is the largest among all edge servers, while the target utilization ratio of other devices is set to 0%, and the array composed of the target utilization ratio of all devices is taken as the initial state; For the improved DRQN algorithm based on LSTM, the time window of LSTM network layer is set to 10; For improved IDQN algorithm based on candidate network, the total size of candidate network set is 20, and Net_1 and Net_2 account for half size of the network set Net, respectively.

5.2. Analysis of experimental results

In order to reflect the resource utilization of mobile applications in different time periods, this paper uses the Google Cluster Trace dataset to simulate changes in the utilization of each module over time [30]. In addition, in order to verify the availability of

Algorithm 3 IDQN Algorithm

Input: Requested batch size δ, current neural network MainNet, candidate network set Net

Output: Trained current neural network

1. Randomly select data samples miniBatch with size δ
2. Initialize state list stateList and reward list rewardList
3. Initialize an array NetTotalArray to store all reward values of Net_2, $Net_2 \in$ Net
4. totalValue = 0
5. N = Size of Net
6. for i=0, δ-1 do
7. targetValue = 0
8. Get current state s_t, the action a_t, the reward r_t and next state s_{t+1} based on the ith sample in miniBatch
9. Calculate the reward list Q_1 of the network MainNet based on the state s_t
10. Calculate the reward list Q_2 of the network MainNet based on the state s_{t+1}
11. Select the action a_{t+1} corresponding to the maximum reward value in Q_2
12. Initialize reward value list NetValueList to store reward values calculated by the candidate network set Net
13. for j=0, N-1 do
14. Calculate the reward value Q_j of the jth network in the Net based on the state s_{t+1} and action a_{t+1}
15. NetValueList[j] = Q_j
16. end for
17. targetValue = Max(NetValueList)
18. $Q_1[a_t]$ = targetValue
19. totalValue = totalValue + r_t
20. stateList[i] = s_t
21. rewardList[i] = Q_1
22. Replace the earliest target network in Net_1 by MainNet every C steps, $Net_1 \in$ Net
23. end for
24. if totalValue > Min(NetTotalArray) do
25. Replace the network with the smallest total reward value in the NetTotalArray with the MainNet, and update the corresponding total reward value as totalValue
26. end if
27. Train the network MainNet based on the stateList and the rewardList
28. Return MainNet

strategies based on deep reinforcement learning algorithms, this paper first selects 1000 applications from the Google dataset to train neural networks, and then selects part of the remaining data to test the trained network model, so as to compare the generality and efficiency of the strategy. Figs. 6 and 7 show the loss function scores of each deep reinforcement learning algorithm in the training process of small batch samples. The smaller the

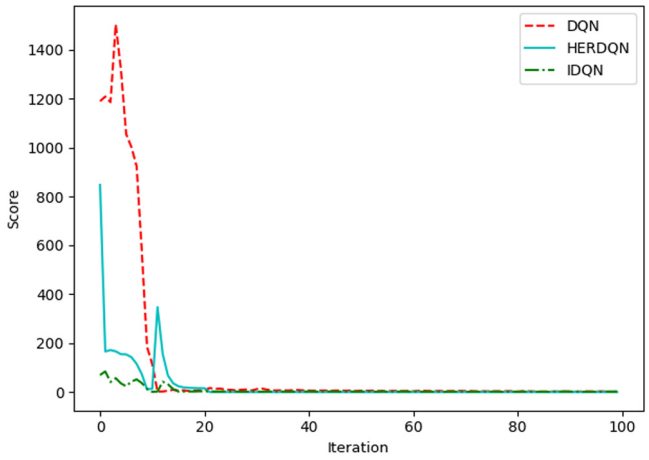


Fig. 6. Iterative figure of loss functions for DQN and IDQN.

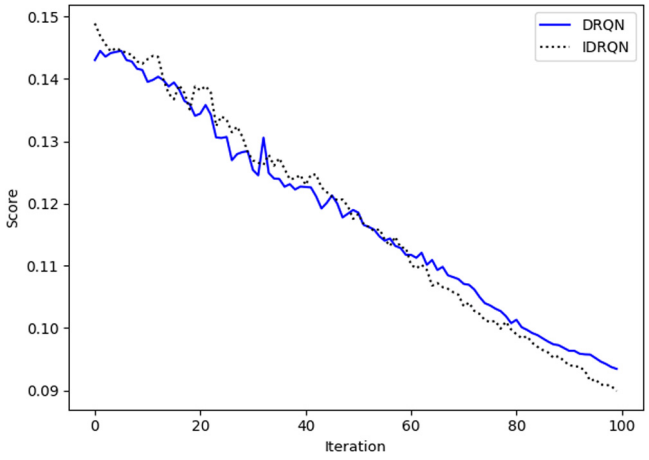


Fig. 7. Iterative figure of loss functions for DRQN and IDRQN.

value of loss function, the better the result of network model. It can be seen from figures that the loss function scores of the IDQN algorithm is lower than the DQN algorithm and HERDQN algorithm in the early iteration period when parameters are the same, which has great advantages for solving the problem of the short training period; The loss function scores of IDRQN algorithm and DRQN algorithm are similar in the early iteration period. However, loss function scores of IDRQN algorithm decrease more than DRQN algorithm as the number of iterations increases, so IDRQN algorithm is easier to approach the optimal solution of the problem when the number of iterations is the same. In addition, the deep reinforcement learning algorithm based on the LSTM network is maintained at a relatively low score during the whole iterative process by comparing the DQN algorithm with the DRQN algorithm and comparing the IDQN algorithm with the IDRQN algorithm. The main reason is that the DRQN algorithm and the IDRQN algorithm can obtain a more comprehensive state by using the LSTM network layer, and the estimated value obtained is closer to the real value.

Fig. 8 is a figure of resource consumption generated by each algorithm in task offloading. It can be seen from the figure that as the number of applications increases, the offloading strategy generated by each algorithm is basically incremental in terms of energy consumption, cost, load balancing, latency, network usage, and execution time. The offloading strategy based on Mobile algorithm can achieve good results in latency and network usage, but the performance of cost and load balancing is in general, and it is the worst of all algorithms in terms of energy consumption and execution time. This is mainly because the Mobile algorithm prefers to offloads subtasks to the local device for execution. When resources of the local device are insufficient, subtasks are gradually offloaded to upper devices. Since some subtasks can be executed locally without network transmission, so the Mobile algorithm has lower network latency and network usage. At the same time, it can be seen that the processing capacity and PPR of mobile devices are much less than remote servers according to Table 2, so processing subtasks on local devices will result in longer execution time and higher energy consumption. In addition, the offloading strategy based on Edge algorithm performs well in load balancing and network usage, but it performs generally in other aspects. The main reason is that the Edge algorithm prefers to offloads subtasks to the edge server cluster, which makes the resource utilization of all edge servers maintain at an average level and minimizes the value of load balancing. At the same time, the performance of edge servers can satisfy the processing requirement of more subtasks, so the network transmission between subtasks is reduced, and the network usage of the whole cluster is further reduced.

The DQN algorithm, HERDQN algorithm, DRQN algorithm, IDQN algorithm, and IDRQN algorithm all use the deep reinforcement learning to automatically generate the corresponding offloading strategy from the value iteration. From results in Fig. 8, it can be seen that as the number of applications increases, the offloading strategy generated by the DQN algorithm performs badly in terms of load balancing, and in other respects it performs generally; The HERDQN performs a little better than the DQN algorithm in all aspects except cost and latency; The DRQN algorithm has good effect on energy consumption and execution time, and the performance is general in terms of cost, latency, and network usage. When the number of applications is large, the strategy generated by IDQN algorithm has good results in terms of cost and latency, but its result is the worst in terms of network usage. The strategies generated by the above three deep reinforcement learning algorithms have their own advantages and disadvantages, and the improved IDRQN algorithm based on LSTM network and candidate network comprehensively considers the advantages of DRQN algorithm and IDQN algorithm in many aspects. The performance of IDRQN algorithm in terms of energy consumption, load balancing and execution time is consistent with DRQN algorithm. When the number of applications is large, the latency of IDRQN algorithm and IDQN algorithm show a downward trend, and its performance is better than the other two deep reinforcement learning algorithms. In addition, IDRQN algorithm and IDQN algorithm have similar performance in terms of network usage, and the result is next only to the heuristic algorithm.

In order to verify the relationship between CPU utilization and resource consumption in heterogeneous devices, this paper uses offloading strategies generated by each algorithm to offload 100 applications, then the impact of types and quantities of heterogeneous devices on resource consumption are analyzed. Table 4 is the number distribution of heterogeneous devices based on CPU utilization, it shows that the strategy generated by IDRQN algorithm and DRQN algorithm tends to offload subtasks to edge servers of model DL360 Gen10, and IDRQN algorithm is the least used in all algorithms for edge servers of model RX350 S7. It can be seen from Table 2 that when the power consumption is the same, the DL360 Gen10 and RX350 S7 are the best and the worst edge server models in terms of energy consumption, respectively. In addition, IDRQN algorithm, DRQN algorithm, and Edge algorithm rarely offload subtasks to local devices with low PPR but zero cost. Therefore, the offloading decisions of these

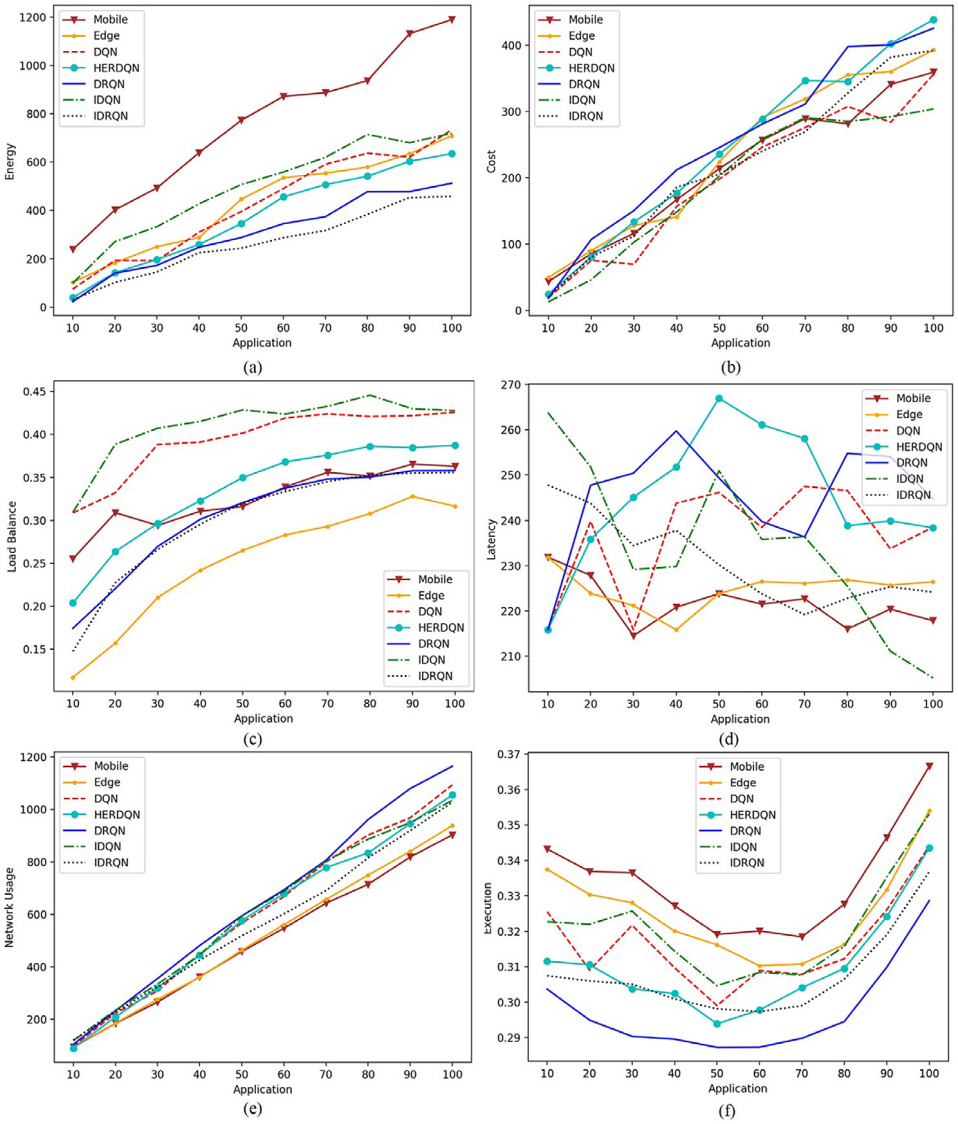


Fig. 8. Resource consumption generated by each algorithm in task offloading.

algorithms generate lower energy and execution time, but they perform generally in terms of cost and network usage. Beyond that, because these decisions generated by the IDQN algorithm and the Mobile algorithm prefer to offload subtasks to local devices, and the IDQN algorithm is more efficient than Mobile algorithm in using edge servers with higher PPR. On the one hand, the performance of the IDQN algorithm in energy consumption is better than that of the Mobile algorithm, but is worse than other algorithms; On the other hand, the IDQN algorithm and the Mobile algorithm perform well in terms of cost, latency, and network usage, while being relatively bad in average execution time. It can be seen from Table 4 that HERDQN algorithm is more effective than DQN algorithm in the use of high PPR edge servers. However, HERDQN algorithm makes poor use of local devices, so it is weaker than DQN algorithm in terms of cost and latency, but slightly better than DQN algorithm in other aspects.

6. Conclusion

In this paper, deep reinforcement learning is proposed to solve the problem of multi-service nodes in large-scale heterogeneous MEC and multi-dependence in mobile tasks. Then the offloading strategy generated by each algorithm is simulated on the edge computing simulation platform iFogSim. Finally, the advantages and disadvantages of each algorithm are verified by comparing various factors such as energy consumption, cost, load balancing, latency, network usage and average execution time. According to the multi-faceted results of each algorithm, the improved IDRQN algorithm based on LSTM network and candidate network can satisfy the latency sensitivity application constructed in this paper to a large extent. In addition, this paper uses various algorithms to offload a certain number of applications and compare the relationship between the number distribution of heterogeneous devices and each resource consumption, so as to prove that the strategy generated by IDRQN algorithm is scientific and effective in solving the task offloading problem of MEC. In the future, we may apply transfer learning to large-scale heterogeneous MEC problems involving various types of mobile applications, so as to improve the training speed and performance between different applications [31,32].

Table 4
Number distribution table of heterogeneous devices with different CPU utilization.

	Mobile	Edge	DQN	HERDQN	DRQN	IDQN	IDRQN
RX4770 M4 [80%,100%]	0	0	0	0	0	0	0
RX350 S7 [80%,100%]	7	8	11	12	1	9	0
DL325 Gen10 [80%,100%]	0	4	8	10	6	10	6
DL360 Gen10 [80%,100%]	1	3	8	9	18	5	18
TX120 [80%,100%]	16	0	6	0	0	16	0
TX150 S5 [80%,100%]	7	0	5	0	0	11	0
TX150 S6 [80%,100%]	7	0	0	0	0	6	0
RX4770 M4 (20%,80%)	0	0	0	0	0	0	0
RX350 S7 (20%,80%)	11	10	1	0	0	1	0
DL325 Gen10 (20%,80%)	15	12	1	0	0	0	1
DL360 Gen10 (20%,80%)	11	14	0	0	0	0	0
TX120 (20%,80%)	2	0	0	0	0	6	0
TX150 S5 (20%,80%)	5	0	0	0	0	4	0
TX150 S6 (20%,80%)	15	0	21	0	0	24	0
RX4770 M4 [0%,20%]	1	1	1	1	1	1	1
RX350 S7 [0%,20%]	2	2	8	8	19	10	20
DL325 Gen10 [0%,20%]	5	4	11	10	14	10	13
DL360 Gen10 [0%,20%]	8	3	12	11	2	15	2
TX120 [0%,20%]	22	40	34	40	40	18	40
TX150 S5 [0%,20%]	18	30	25	30	30	15	30
TX150 S6 [0%,20%]	8	30	9	30	30	0	30
Quantity of [80%,100%]	38	15	38	31	25	57	24
Quantity of (20%,80%)	59	36	23	0	0	35	1
Quantity of [0%,20%]	64	110	100	130	136	69	136
Quantity of devices	161	161	161	161	161	161	161

Acknowledgment

The work was supported by the National Natural Science Foundation (NSF) under grants (No.61472139)

Declaration of competing interest

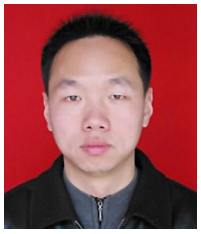
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[2] C. You, K. Huang, H. Chae, B.H. Kim, Energy-efficient resource allocation for mobile-edge computation offloading, IEEE Trans. Wireless Commun. 16 (3) (2017) 1397–1411.
[3] C. Wang, C. Liang, F.R. Yu, Q. Chen, L. Tang, Computation offloading and resource allocation in wireless cellular networks with mobile edge computing, IEEE Trans. Wireless Commun. 16 (8) (2017) 4924–4938.
[4] Changsheng You, Kaibin Huang, Hyukjin Chae, et al., Energy-efficient resource allocation for mobile-edge computation offloading, IEEE Trans. Wireless Commun. 16 (3) (2016) 1397–1411.
[5] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, Y. Zhang, Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks, IEEE Access 4 (2016) 5896–5907.
[6] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2016) 2795–2808.
[7] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1628–1656.
[8] Y. Mao, J. Zhang, S.H. Song, et al., Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems, IEEE Trans. Wireless Commun. 16 (9) (2017) 5994–6009.
[9] T.Q. Dinh, J. Tang, Q.D. La, T.Q. Quek, Offloading in mobile edge computing: Task allocation and computational frequency scaling, IEEE Trans. Commun. 65 (8) (2017) 3571–3584.
[10] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Areas Commun. 34 (12) (2016) 3590–3605.
[11] H.R. Arkian, A. Diyanat, A. Pourkhalili, MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications, J. Netw. Comput. Appl. 15 (82) (2017) 152–165.
[12] L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, Cost efficient resource management in fog computing supported medical cyber-physical system, IEEE Trans. Emerg. Top. Comput. 5 (1) (2017) 108–119.
[13] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, Optimized IoT service placement in the fog, Serv. Orient. Comput. Appl. 11 (4) (2017) 427–443.
[14] L. Yang, J. Cao, G. Liang, X. Han, Cost aware service placement and load dispatching in mobile cloud systems, IEEE Trans. Comput. 65 (5) (2016) 1440–1452.
[15] F. Messaoudi, A. Ksentini, P. Bertin, On using Edge Computing for computation offloading in mobile network, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–7.
[16] J. Ren, G. Yu, Y. Cai, Y. He, F. Qu, Partial offloading for latency minimization in mobile-edge computing, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–6.
[17] J. Liu, Q. Zhang, Offloading schemes in mobile edge computing for ultra-reliable low latency communications, IEEE Access 6 (2018) 12825–12837.
[18] C.F. Liu, M. Bennis, H.V. Poor, Latency and reliability-aware task offloading and resource allocation for mobile edge computing, in: 2017 IEEE Globecom Workshops (GC Wkshps), 2017, pp. 1–7.
[19] Q. Zheng, R. Li, X. Li, J. Wu, A multi-objective biogeography-based optimization for virtual machine placement, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2015, pp. 687–696.
[20] J. Zhang, X. Hu, Z. Ning, et al., Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, IEEE Internet Things J. 5 (4) (2017) 2633–2645.
[21] L. Huang, X. Feng, C. Zhang, et al., Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing, Digit. Commun. Netw. 5 (1) (2019) 10–17.
[22] M. Adhikari, T. Amgoth, Heuristic-based load-balancing algorithm for IaaS cloud, Future Gener. Comput. Syst. 81 (2018) 156–165.
[23] C. Guerrero, I. Lera, C. Juiz, A lightweight decentralized service placement policy for performance optimization in fog computing, J. Ambient Intell. Humaniz. Comput. (2018) 1–18.
[24] Z.X. Xu, et al., Deep reinforcement learning with sarsa and Q-learning: A hybrid approach, IEICE Trans. Inf. Syst. E101d (9) (2018) 2315–2322.
[25] L. Teng, T. Bin, A. Yun, et al., Parallel reinforcement learning: a framework and case study, IEEE/CAA J. Autom. Sin. 5 (4) (2018) 827–835.
[26] L. Lepetit, F. Strobel, Bank insolvency risk and time-varying Z-score measures, J. Int. Financ. Mark. Inst. Money 25 (2013) 73–87.
[27] W. Yu, R. Wang, R. Li, et al., Historical best Q-networks for deep reinforcement learning, in: 2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2018.
[28] M. Andrychowicz, F. Wolski, A. Ray, et al., Hindsight experience replay, Adv. Neural Inf. Process. Syst. (2017) 5048–5058.
[29] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, et al., IFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Softw. - Pract. Exp. 47 (9) (2017) 1275–1296.
[30] C. Reiss, J. Wilkes, J.L. Hellerstein, Google cluster-usage traces: format+ schema. Google Inc. White Paper, 2011: 1-14.
[31] S. Wang, C. Tang, J. Sun, et al., Cerebral micro-bleeding detection based on Densely connected neural network, Front. Neurosci. 13 (2019) 422.
[32] S.H. Wang, S. Xie, X. Chen, et al., Alcoholism identification based on an AlexNet transfer learning model, Front. Psychiatry (2019) 10.

Haifeng Lu was born in 1993. He graduated from computer science department of Donghua University in Shanghai in 2017 and obtained his master's degree. In the same year, he studied in the school of information science and engineering of East China University of Science and Technology and studied for his doctorate degree. His main research interests are edge computing and reinforcement learning.

Chunhua Gu was born in 1970. Professor and Ph.D. supervisor in the School of Information Science and Engineering, East China University of Science and Technology. Senior member of China Computer Federation. His main research interests include cloud computing and internet of things.



Fei Luo was born in Wuhan, Hubei Province, China in 1978. He received the B.S., M.S. and Ph.D. degrees in Computer Science from Huazhong University of Science and Technology in 1997, 2004 and 2008, respectively. From 2008 to 2015, he was an Assistant Professor in the College of Information Science and Engineering, East China University of Science and Technology, and he have been an Associate Professor since 2015 in the same college. He is the author of more than 30 papers and more than 10 inventions. His research interests include distributed computing and cloud computing.

Weichao Ding was born in 1989. Ph.D. candidate in the School of Information Science and Engineering, East China University of Science and Technology. He received the B.S. degree in computer science and technology from Northeast Forestry University, Haerbin, China, in 2013 and is currently pursuing the M.S and Ph.D. degree in computer applications at East China University of Science and Technology, Shanghai, China. His main research interests include cloud computing, cloud resource management and optimization, big data applications.

Xinping Liu was born in 1993. He studied in the school of information science and engineering of East China University of Science and Technology and studied for his master degree. His main research interests are multi-objective evolutionary algorithms and cloud resource scheduling.