Future Generation Computer Systems 129 (2022) 64–76

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Response time and energy consumption co-offloading with SLRTA algorithm in cloud–edge collaborative computing

Zhao Tong [a,*], Xiaomei Deng [a], Jing Mei [a], Bilan Liu [a], Keqin Li [b]

[a] College of Information Science and Engineering, Hunan Normal University, Changsha, 410012, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 29 March 2021
Received in revised form 31 August 2021
Accepted 16 November 2021
Available online 29 November 2021

Keywords:
Deep reinforcement learning (DRL)
Mobile cloud–edge collaborative
Resource allocation
Task offloading

ABSTRACT

Mobile edge computing (MEC) has emerged as a computing model that provides services near data generation sources. However, although MEC servers have more computing capability and resources than local, they cannot support the processing of too many user equipment (UE). In contrast, cloud servers have at least several times the computing capability and resources of MEC servers, so they can perform more tasks in parallel and with faster processing speeds. Therefore, the cloud–edge collaborative computing model combines respective advantages of MEC and cloud, and it is thus highly suitable for processing various types of tasks. How to offload tasks to the computing nodes efficiently and allocate resources optimally is an important research question. In this paper, we propose a task offloading and resource allocation algorithm with resource and reliability constraints in the cloud–edge collaborative computing environment. Three computing nodes can be used to process the task data, including UE, the MEC server and the cloud server, and the computing nodes are constrained by their computing capability, resources, and reliability. An online learning algorithm based on deep reinforcement learning (DRL) is designed for the objective optimization problem. Through the evaluation and verification of simulation experiments, the proposed algorithm is shown to be capable of effectively improving the quality of the user experience and reducing the energy consumption of the system.

1. Introduction

With the high-speed progress development of the cyber–physical systems (CPS) and wireless communication networks, user smart terminal equipment is generating a great quantity of task data. According to the statistics, global smart devices will produce 50ZB of data in 2020, an increase of about 25% compared with 2019. And such data production will continue to increase in the future. Such a large amount of data poses a huge challenge to the traditional cloud computing model. To reduce response delays, reduce network loads, improve user service experiences (USE), etc. Inline with the development of the times, edge computing is used to specifically process data at the edge of a network [1–4]. Mobile edge computing (MEC) [5,6] is a specific implementation mode in edge computing, and the advent of 5th-generation (5G) mobile networks has mostly benefited from the proposal of MEC [7]. The MEC model has a better effect than the cloud computing model in processing real-time, high-response tasks. However, due to the limitation of the edge intelligent equipment ability. The computing process and storage capacity of MEC servers are relatively weak. The MEC server has insufficient capability and resources when processing computation-intensive tasks, and such tasks are more suitable for processing with the cloud computing model. In order to better serve users, different computational paradigms are needed to deal with them. MEC is suitable for processing real-time and small data tasks; cloud computing is suitable for processing non-real-time and long-cycle computation-intensive tasks.

Average response time and total energy consumption are two very important metrics. Shorter average response times can make the user experience better. The user wants the response time to be as short as possible. Lower energy consumption can lead to more cost savings for service providers. Service providers want to minimize overall energy consumption as much as possible. Obviously, these two metrics are opposites. To efficiently process multiple types of task data, the use of the mobile cloud–edge collaborative computing model to process tasks is considered. The mobile cloud–edge collaborative computing model is the complement and optimization of MEC and cloud computing, mainly including the coordination of computations, resources, data, and management. It combines centralized and distributed processing methods to maximize the overall computing advantages. At

* Corresponding author.
E-mail address: tongzhao@hunnu.edu.cn (Z. Tong).

https://doi.org/10.1016/j.future.2021.11.014
0167-739X/

present, the research directions with respect to the MEC or cloud–edge collaborative computing environments mainly include task offloading and resource allocation, big data analysis, intelligent applications, privacy, and security [8–10]. Among them, task offloading and resource allocation are essential hot research topics that play a crucial role in improving service quality [11,12].

In this paper, we propose a self-learning task offloading and resource allocation algorithm (SLTRA) in the mobile cloud–edge collaborative computing environment. This algorithm uses the DRL method to choose an appropriate computing node for the given task and attempts to reduce the task response time and energy consumption under the constraints of system resources and reliability; it is verified through experiments that the proposed algorithm has further improvement than existing methods. The main contributions of this paper are as follows.

- We model the task offloading and resource allocation problem as a Markov decision process (MDP) and propose a solution based on the DRL method; this approach can improve the optimization performance of the algorithm through continuous adaptive learning.
- In order to achieve user's quality of experience (QoE) further needs and reduce task execution costs, the objective problem is defined to minimize the average task response time and the system energy consumption, considering both interests of the user aspect and the service providers aspect.
- When thinking about the DRL method's state and reward designing, we consider the optimization objective through a weighting method to combine the task response time and entire system energy consumption.
- The experiment was performed in a computing environment built on the Python platform, while considering the mobility of user equipment (UE), resource constraints, and the reliability of the computing nodes. Experimental results show an excellent performance.

2. Related work

2.1. Research background

At present, researchers have done a lot of works on task offloading and resource allocation in the cloud or MEC environment, such as [13–15]. Because the cloud–edge collaborative computing architecture is still in the early stage of development, there has been relatively little research done in this computing environment.

Chen [16] proposed a solution based on the semi-definite relaxation and random mapping methods, which can make optimal offloading decisions for users. Liu [17] put forward the task offloading and a resource collaboration algorithm based on queuing theory and defined the multi-objective optimization problem to minimize energy consumption and task delays. Kofler [18] proposed a novel approach that automatically optimizes task partitioning for different problem sizes and different heterogeneous multi-core architectures. Du [19] recommended a sub-optimal method with reducing calculation pressure for computing nodes in the collaborative environment that can save costs of delays and energy consumption and ensure fairness among users. Mukherjee [20] advanced a data offloading strategy to solve task delay sensitive problems, and this strategy mainly considers the trade-off between local execution delays and transmission delays during task offloading. Grewe [21] proposed a portable partitioning scheme for OpenCL programs on heterogeneous CPU–GPU systems, and developed a purely static approach based on predictive modeling and program features. Hao et al. [22] raised an algorithm that can provide personalized task offloading for the cloud–edge collaborative system. The algorithm uses coarse-grained and fine-grained calculation methods, which can meet the service requirements of tasks. Tong [23] came up with a dynamic task scheduling algorithm to maintain machine load balance and reduce task rejection rate, under the premise of meeting the SLA factor. Hu [24] proposed a game-based offloading algorithm, which had a good performance on reducing system energy. Ouyang [25] proposed a cluster-based task offloading method and a backtrack-based trust evaluation mechanism to assist offloading by examining trust scores and energy consumption. Wu [26] considered computing the security performance to ensure task safety with rigorous mathematical way. Yan [27] brought up a deep reinforcement learning framework, quickly choosing superior offloading to minimize its efficiency. Lee [28] presented a VFC resource allocation algorithm, which combined with reinforcement learning. The algorithm performed better than traditional algorithm in real-time data responding in the car networking task scenario. Huang [29] designed a protocol to achieve less conflict for wake-up radio enabled WSNs in data transmission, showing better performance in saving energy and shortening delay.

However, above mentioned algorithms have not considered task offloading strategy in the MEC environment. Most of the task offloading and resource allocation algorithms are based on traditional mathematical theory, and the performances of such algorithms were verified in a numerical solution-based experimental environment, which lacks some realism. In this paper, we use the intelligent learning capability of deep reinforcement learning (DRL) [30,31] to effectively improve algorithmic performance.

2.2. New material

This article is a further study based on our previous work [15]. Compared with the previous article, this article has several new innovations, listed as follows.

- In this paper, we use a three-tier framework model. Compared to the previous one, we add a cloud processing layer for tasks with high computational demand to improve user experience. At the same time, with the complexity of our model, the design of task communication between layers becomes more complicated.
- Because of model changes, in this paper the offloading scheme has been changed, so the optimization algorithm we propose to solve the problem is different. In addition, we also added the probability of processor failure to improve the authenticity of the model. The figure on the left below is the algorithm of the previous article, on the right is the algorithm we proposed in this article.
- The data set we use is also different. In this paper, in order to be closer to reality, we add real data in the experiment section. The data is come from the google dataset https://commondatastorage.googleapis.com/clusterdata-2011-1/.
- In order to be more in line with real life, the possibility of machine failure is considered in this paper. The index "isFail" is to represent the state of every computing node. "isFail=0" means that the computing node is working normally, and can process the allocated task. Otherwise, the computing node is in a failure period, we should reassigned a suitable computing node for the task.
- For different model and algorithm, the trained neural network is also different, and the parameters set to make the neural network converge are different. The hyperparameter setting are as follows, the left is the previous, and the right is in this paper.
- This paper add new experiment at Section 5 to test the performance of the algorithm more comprehensively.

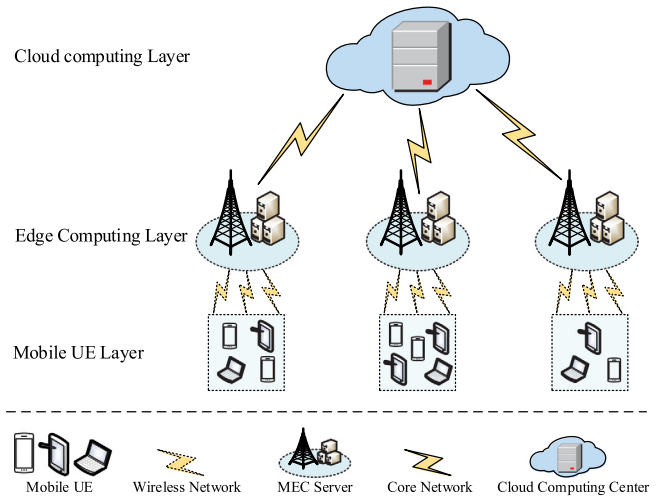


Fig. 1. Mobile cloud–edge collaborative computing model.

The rest of the paper is organized as follows. Section 3 describes the system model and defines the objective problem. Section 4 introduces the algorithm proposed in this paper. Section 5 presents and analyzes the experimental results. Finally, conclude this paper and mention future possible research perspectives.

3. System model and problem formulation

In this section, we first introduce the cloud–edge collaborative computing model. Second, we analyze the response time model and the energy consumption model. Finally, we formulate the definition of the objective problem.

3.1. Cloud–edge collaborative computing model

The three-layer cloud–edge collaborative computing model as shown in Fig. 1. It mainly includes a mobile UE layer, an edge computing layer, and a cloud computing layer. Compared with the two-layer framework on the edge and local sides, the three-layer framework has one more cloud processing layer. The addition of the cloud computing layer makes it more suitable to handle higher computationally intensive tasks. The edge layer could handle tasks not far distributed, and the computational is not exceptionally high. The mobile UE layer only needs to process those tasks with small computational requirements. Each layer has its advantages. The model is composed of N pieces of UE, K MEC servers and 1 cloud server center. The mobile UE generates tasks that are independent of each other, and the time interval for generating the tasks obeys a Poisson distribution [32]. Each task can be run locally, in the cloud, or on the edge. The task offloaded to the edge computing layer should via wireless network transmission. After the MEC server completes the task, it transmits the obtained result back to the UE through the base station (BS). Since the distances between the cloud computing center and UE devices are relatively far, a task sent directly from the UE layer to the cloud computing layer requires a lots of transmission energy. If the task is offloaded to the cloud computing layer for processing, the UE first sends the task data to the base station in the edge computing layer, and then the base station sends the task to the cloud computing layer through the core network. Similarly, after the task is processed completely, the obtained result is returned to the UE through the original path. In this situation, the edge computing layer is used as the transfer station for tasks to be sent from UE devices to the cloud computing center.

3.2. Response time model

The task response time is the time interval from when the task is submitted to moment when the task result is returned, and it consists of the task waiting time, transmission time, and processing time. If the task is processed on a local UE, we treat the communication time as 0. In the sub-section, we calculate the response times of the task on different computing layers and obtain the average response time for tasks.

3.2.1. Local computing

The local computing mode refers to a task being processed on the local UE that generates the task data. The response time is consisted of the task in the waiting queue and the processing time. If the UE is idle, the waiting time for the task is 0; otherwise, the task needs to wait until the previous task is completed. The calculation of the response time res_i for task i processed on UE u is defined as follows:

$$t^s_{i,u} = \begin{cases} sub_i, & \text{UE } u \text{ is idle;} \\ avail_u, & \text{otherwise;} \end{cases} \tag{1}$$

$$t^r_{i,u} = t^s_{i,u} + \frac{d_i c_i}{f_u}, \tag{2}$$

$$res_i = t^r_{i,u} - sub_i, \tag{3}$$

where $t^s_{i,u}$ is the time when the execution of task i on UE u starts, and $t^s_{i,u}$ has two situation. When UE is idle, $t^s_{i,u}$ is the submission time; otherwise, $t^s_{i,u}$ is the available time for the UE u. $t^r_{i,u}$ is the time required for a result to be returned, d_i is the data size, c_i is the number of CPU cycles needed to compute one bit of data, and f_u is the CPU frequency of UE u.

3.2.2. Edge computing

The edge computing mode refers to offloading a task to the edge computing layer for processing. The response time is the sum of the waiting time, transmission time, and processing time. If the MEC server allocated to the task has sufficient computing resources, the task does not need to wait for resources to be released, the waiting time is 0. Otherwise, the execution of the task must wait for the server to acquire sufficient resources. Since the data size of the task result is small compared to that of the task itself, the transmission time here refers to the time cost of offloading task data from the UE to the base station in the edge computing layer, and the time cost of returning from the base station to the UE is negligible. The transmission rate [33,34] of UE u to base station m is defined as:

$$r_{u,m} = W \log_2\left(1 + \frac{p_u g_{u,m}}{N_0 W}\right), \; g_{u,m} = \left(D_{u,m}\right)^{-\sigma}, \tag{4}$$

where W, $g_{u,m}$ and $D_{u,m}$ are the communication bandwidth, channel gain and distance between UE u and base station m, respectively. p_u is the transmission power of UE u, N_0 is the noise power spectral density of the base station m, and σ is the path-loss exponent.

The calculation of the response time res_i for task i processed on MEC server j of base station m is defined as follows:

$$t^s_{i,m_j} = \begin{cases} sub_i + \frac{d_i}{r_{u,m}}, & \text{resources are sufficient;} \\ sub_i + \frac{d_i}{r_{u,m}}, & wait_i < trans_i; \\ avail_{m_j}, & \text{otherwise;} \end{cases} \tag{5}$$

$$t^r_{i,m_j} = t^s_{i,m_j} + \frac{d_i c_i}{f_{m_j}}, \tag{6}$$

$$res_i = t^r_{i,m_j} - sub_i, \tag{7}$$

where t^s_{i,m_j} is the time when the execution of task i on MEC server j starts, and t^s_{i,m_j} has three situation. If the server has sufficient resources, t^s_{i,m_j} is the sum of the submission time and transmission time; if the server resources are insufficient but the waiting time $wait_i$ for the resources to be released is less than the transmission time $trans_i$, t^s_{i,m_j} is the sum of the submission time and transmission time; otherwise, t^s_{i,m_j} is the available time after server j has sufficient resources, and this is defined as $avail_{m_j}$. t^r_{i,m_j} is the time required to return the result of task i, and f_{m_j} is the CPU frequency of MEC server j of base station m.

3.2.3. Cloud computing

The cloud computing mode refers to offloading a task to the cloud computing layer for processing. Because the cloud server has sufficient resources, there is no need to wait other tasks' execution. The response time is consist of the transmission time and the processing time. The transmission time includes the task data delivered from the local equipment to the base station, the time from the base station to the cloud center, and the time required to transmit the result data from the cloud center to the base station. Because of the local equipment is far from the cloud, the result transmission time between the base station and the cloud center is considered. The calculation of the response time res_i for task i processed on the cloud server is defined as follows:

$$t^s_{i,c} = sub_i + \frac{d_i}{r_{u,m}} + \frac{d_i}{r_{m,c}}, \tag{8}$$

$$t^r_{i,c} = t^s_{i,c} + \frac{d_i c_i}{f_c} + \frac{d^r_i}{r_{m,c}}, \tag{9}$$

$$res_i = t^r_{i,c} - sub_i, \tag{10}$$

where $t^s_{i,c}$ is the time when the execution of task i on the cloud server starts, and $r_{m,c}$ is the transmission rate between the base station and the cloud computing center. $t^r_{i,c}$ is the time required to return the result of task i, f_c is the CPU frequency of the cloud server, and d^r_i is the result data's size.

The response time of the task is obtained according to Eqs. (1)–(10), but the response time of a single task can only reflect the specific processing situation. Therefore, we use the average response time to reflect the overall performance situation, as doing so can reflect the user's QoE level. The average response time for processing n tasks in the system is defined as follows:

$$res_{avg} = \sum_{i=1}^{n} \frac{res_i}{n}. \tag{11}$$

3.3. Energy consumption model

Energy consumption is the primary indicator of energy conservation and consumption reduction. Corresponding to the task response time model, the energy consumption considered in this paper mainly includes transmission energy and the energy consumed for calculation purposes, and the energy consumed while a task is waiting for execution is neglected. In the sub-section, we calculate the energy consumption of the task at different computing layers and the entire system.

3.3.1. Local computing

The local computing model only considers the energy consumed when computing tasks, and the energy consumption of task i processed on UE u is defined as follows:

$$P_{i,u} = \kappa \left(f_u\right)^3, \tag{12}$$

$$E_i = P_{i,u} \frac{d_i c_i}{f_u}, \tag{13}$$

where $P_{i,u}$ is the power consumption of task i on UE u, and κ is the effective switching capacitance in the chip.

3.3.2. Edge computing

The energy consumption for offloading a task to the MEC server for processing includes the energy consumption for the transmission of task data from the UE to the base station and the energy consumption incurred by calculations on the server. The energy consumption of task i processed on server j of base station m is defined as follows:

$$E_i = p_u \frac{d_i}{r_{u,m}} + d_i q_{m_j}, \tag{14}$$

where q_{m_j} is the energy consumption of server j on base station m when calculating one bit of task data.

3.3.3. Cloud computing

According to the path of task uploading and returning the result, the energy consumption of a task that is offloaded to the cloud server for execution includes the energy consumption of transmission from the UE to the cloud computing center, the energy consumption incurred by calculations, and the energy consumption of returning the results from the cloud center. The energy consumption of offloading a task i generated by UE u to the cloud server is defined as:

$$E_i = p_u \frac{d_i}{r_{u,m}} + p_m \frac{d_i}{r_{m,c}} + d_i q_c + p_m \frac{d_{i,r}}{r_{m,c}}, \tag{15}$$

where p_m is the transmission power of the base station m and the cloud computing center, and q_c is the energy consumption required for the cloud server to calculate one bit of data.

In this paper, we use the total energy consumption of all tasks as the evaluation indicator of energy consumption for the system, and this indicator is defined as follows:

$$E_{total} = \sum_{i=1}^{n} E_i. \tag{16}$$

3.4. Problem formulation

The purpose of this paper is to optimize task offloading and resource allocation problem in a mobile cloud–edge collaborative computing environment while considering the resource constraints and reliability of computing nodes. These limitations make the computing environment close to realistic application environments. The computing nodes include UE devices, MEC servers, and cloud servers center. The computing capabilities and resources of the three types of computing nodes are listed in increasing order. The optimization objective of this paper is to minimize the average response time and total energy consumption under the resource and reliability constraints of the computing nodes. Since UE devices are the generators of task data, the reliability constraints only consider to the MEC servers and the cloud server. First, a computing node that minimizes the objective function is selected for the task. Second, whether the selected computing node meets the resource and reliability requirements is determined. If one of these requirements are not met, the computing node is re-allocated for the task, until the task is allocated to the appropriate computing node. The objective

optimization problem is defined as:

$$\begin{aligned} \min_{\{y\}} \quad & res_{avg} \text{ and } E_{total}, \\ \text{s.t.} \quad & y \leqslant \delta, \\ & \sum_{i=1}^{y} mem_i \leqslant M, \\ & \sum_{i=1}^{y} cpu_i \leqslant C, \\ & isFail = 0, \end{aligned} \tag{17}$$

where y and δ are the number of tasks waiting to be executed in parallel, and the maximum waiting task queue value in parallel on the computing node, respectively. mem_i and cpu_i are the memory and CPU resources required for computing task i, respectively. M and C are the memory and CPU resource capacities of the computing node, respectively. The value of *isFail* represents whether the MEC server or cloud server assigned to the task is in a failure state, and if the task is executed on the UE, the constraint of this condition is not considered. $isFail = 0$ means that the computing node is not in a failure period, and can process the allocated task. Otherwise, $isFail \neq 0$ means that the computing node is in a failure period, it cannot processed any task, so a suitable computing node needs to be re-assigned for the task.

4. Online learning algorithm design

4.1. Algorithm design

The objective in this paper is essentially a decision-making problem: making decisions with respect to task offloading and resource allocation. The Q-learning (QL) algorithm in the field of reinforcement learning has a good effect on solving the decision-making problem, and it is an unsupervised learning algorithm that does not require external guidance, thereby achieving the purpose of repeated learning according to its "trial and error" interactions with the environment [35,36]. The basic elements of QL include an agent, a state, an action, and a reward. The agent continuously shifts from one state to another through the corresponding action for exploration, and the environment feeds back the reward value responding the follow-up impact after taking this action. During this process, the agent obtains a profit value (Q-value) under each state–action pair, thus learning autonomously and making decisions in the correct direction. The update process for the Q-value is defined as follows:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\left(r + \gamma \max_{a'} Q_t\left(s', a'\right) - Q_t(s,a)\right), \tag{18}$$

where s and s' are the current environment and the next environment after taking a specific action, respectively. a and a' are the actions made in the current state and the new state, respectively. α is the learning rate, and γ is the discount factor.

In the QL algorithm, a two-dimensional matrix (Q-table) is used to store Q-values. When the scale of the problem increases, the state space and action space are large, the Q-table cannot store a large number of Q-values. The increase of dimensions makes the solution very difficult. Therefore, Deepmind considered combining QL with deep learning in [37] and proposed the concept of deep Q-network (DQN). In DQN, a convolutional neural network (CNN) [38] is used to fit the update of the Q-table, and this can automatically extract complex features and achieve a good learning effect. This paper intends to use DQN to make optimal decisions for scheduling. The state, action, and reward are set as follows:

1. The method of setting the state is defined in Eq. (19), which is the weighted sum of the normalized response time and normalized energy consumption for the task on the computing node, and the states of all computing nodes constitute the state space. To reduce the influence of subjective factors on the algorithm, the entropy weight method [39] is used in the weighting process.

$$s_i = w_{i,1} res_i' + w_{i,2} E_i', \tag{19}$$

   where $w_{i,1}$ and $w_{i,2}$ are the weights of response time and energy consumption respectively.
2. An action is defined as selecting the computing node for the allocated task, and the number of actions is the number of computing nodes. When the computing node is selected, the corresponding action value is setting for 1, and the other unselected action values are setting for 0. Assume that the order of the computing nodes in the action space is as follows: UE, MEC servers, and cloud server. If tasks are offloaded to the cloud, the action space is setting as $(0, 0, \ldots, 1)$.
3. The reward value of environmental feedback is beneficial for guiding the learning of the agent, and the setting of the reward function has an important influence on the learning effect. A value setting that is too large may lead to overfitting, and a value setting that is too small may lead to too slow convergence. Since the goal of this model is to jointly optimize task response time and energy consumption, we consider the degree of objective minimization as the basis for setting the reward value, which is defined as follows:

$$r = -\left(w_{i,1} res_i' + w_{i,2} E_i'\right). \tag{20}$$

Lots of generating tasks are queued to allocate computing resources according to the generation time. This means that tasks cannot be preempted. The algorithm proposed in this paper makes decisions on whether to offload each task and which computing node to offload the task to. With the increase of processing task numbers, the accuracy of the computing node selection process based on intelligent algorithms also increases, and this can reduce the task response time and energy consumption of the system. First, the algorithm is used to make the decision of selecting a computing node. The computing node is a UE, MEC server or cloud server. Next, consider whether the computing node assigned to the task has sufficient resources and can work properly. If the existing resources of the computing node cannot fully meet the resource requirements of the task, the task needs to wait for a release of resources. Considering whether the computing node can work normally, if the MEC server or cloud server selected to process the task is broken down, an MEC server with sufficient computing resources that can work normally is reallocated, and the reassigned server is now located on the base station to which the task belongs.

We consider reassigning computing nodes for a given task because the recovery time required for a machine failure is usually longer than the transmission and computation time of the task. The pseudocode of the SLTRA algorithm is as follows:

4.2. Hyperparameter tuning

The value of the hyperparameter has an important influence on the result. These two parameters directly affect whether the algorithm can attain convergence or not and the speed of convergence. Therefore, in this section, we conduct comparative experiments on the two hyperparameters of learning rate α and discount factor γ to obtain appropriate value making the algorithm perform better. The experimental result are shown in Figs. 2 and 3.

Algorithm 1: The SLTRA Algorithm

Input: Mobile user equipment
Output: Average response time, total energy consumption
Initialize the computing environment and parameters;
for *each task generated by UE u* **do**
  Use the DQN method to select a computing node;
  if *the selected computing node is UE u* **then**
    The task is processed locally;
  else if *the selected computing node is an MEC server* **then**
    if *the MEC server is broken down* **then**
      Offload the task to the MEC server with the most resources and no failures;
    else
      Offload the task to the selected server;
  else
    if *the cloud server is broken down* **then**
      Offload the task to the MEC server with the most resources and no failures;
    else
      Offload the task to the cloud server;
  Calculate the response time and energy consumption;
  Update the optimal strategy;
end

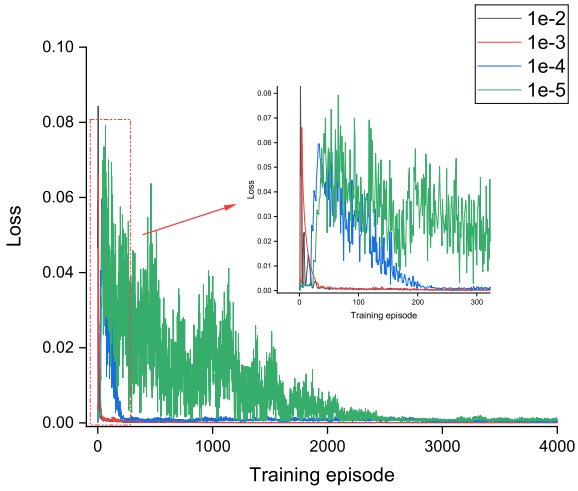


Fig. 2. Convergence of different learning rates.

In Fig. 2, we can apparently see all of these four learning rates can eventually converge successfully, while from the partially enlarged picture, it is obvious that the convergence speed is the fastest when the learning rate is 1e−2 and 1e−3, the third is 1e−4, and the slowest is 1e−5. From the perspective of curve oscillation, the most stable is 1e−3, and the worst is 1e−5. Considering these two aspects, we choose 1e−3 as the value of learning rate α. That is because too large a learning rate may cause the algorithm to fail to converge, while a too small learning rate will cause the learning speed to be too slow and affect the convergence speed.

The discount factor reflects the preference for immediate returns. The discount factor reflects the degree of emphasis on future returns. The larger the value, the more critical the future returns. It can be seen from Fig. 3 that when the discount factor is 0.6, the algorithm converges best, and the second is 0.8.

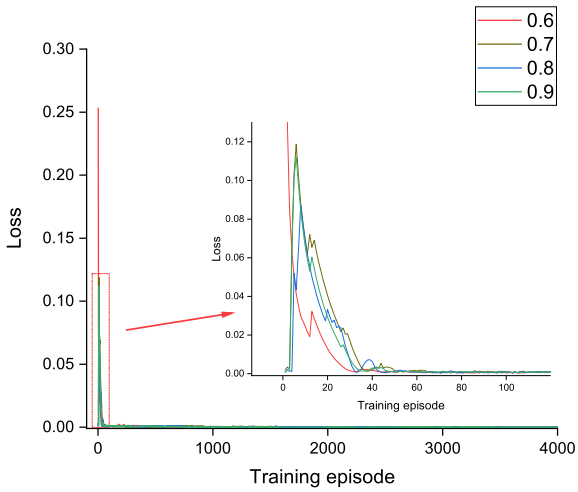


Fig. 3. Convergence of different discount factors.

5. Experiments and analysis

In this section, we compare and analyze the optimization performance of the proposed algorithm under different experimental environment settings. We build the experimental simulation environment on the Python 3.6 platform with the Windows 10 operating system. The simulation parameters is shown in Table 1.

We use the following algorithms to conduct the performance comparison with the algorithm proposed in this paper. Each of these algorithms has the following characteristics: **QL** is a classic reinforcement learning algorithm with good self-learning ability. The **round robin (RR)** algorithm considers assigning tasks to computing nodes in turn for processing. On the basis of the RR algorithm, the **weighted round robin (WRR)** algorithm allocates using different weights according to the power of every computing node, and this is done to balance the computing capabilities of the various computing nodes. **The random** algorithm randomly assigns task requests to computing nodes for processing.

5.1. Proportion of tasks

In the three-layer mobile cloud–edge system, there are three types of computing nodes: a mobile UE, an MEC server and a cloud server. A tasks can be assigned to any of them as a computing platform. Therefore, these three types of computing nodes have a competitive relationship with regard to task processing. In different experimental environments, each computing layer may have different competitive advantages, resulting in changes in the proportion of the number of tasks executed on different computing layers. For example, the UE layer has relatively weak computing power, but no communication delay. Cloud server has strong computing capacity, but it is difficult to meet the real-time demand for tasks.

To reflect the collaborative competition between the computing layers when processing tasks, we consider a statistical experiment regarding the proportions of executed tasks under different data sizes. The data sizes are 100–500 KB, 500–1000 KB, 1000–3000 KB, 3000–5000 KB, 5000–8000 KB and 8000–10000 KB. The experimental results are shown in Fig. 4. According to the results, with the task size increasing, the number of tasks executed on the UE fluctuates little and is stable. The number of tasks executed on the MEC server is constantly decreasing, and correspondingly, the number of tasks executed on cloud servers shows an increasing trend. Therefore, when the calculation pressure of the three-layer computing platform increases, the ability of the MEC server to

Table 1
Experimental simulation parameters.

Notation	Description	Value
$D_{u,m}$	the distance between UE u and base station m	randint (50, 200) m
N_0	the noise power	−174 dBm/Hz
W	the communication bandwidth	2.0 MHz
c_i	the number of CPU cycles needed to compute one bit of data	500 cycles/bit
f_c	the CPU frequency of the cloud server	20 GHz
f_{m_j}	the CPU frequency of MEC server j of base station m	Unif (5.0, 10.0) GHz
f_u	the CPU frequency of UE u	Unif (0.5, 1.0) GHz
p_u	the transmission power of UE u	100 mW
p_m	the transmission power of the base station m and the cloud computing center	200 mW
$r_{m,c}$	the transmission rate between the base station and the cloud computing center	25 mbps
κ	the effective switching capacitance in the chip	10^{-28}

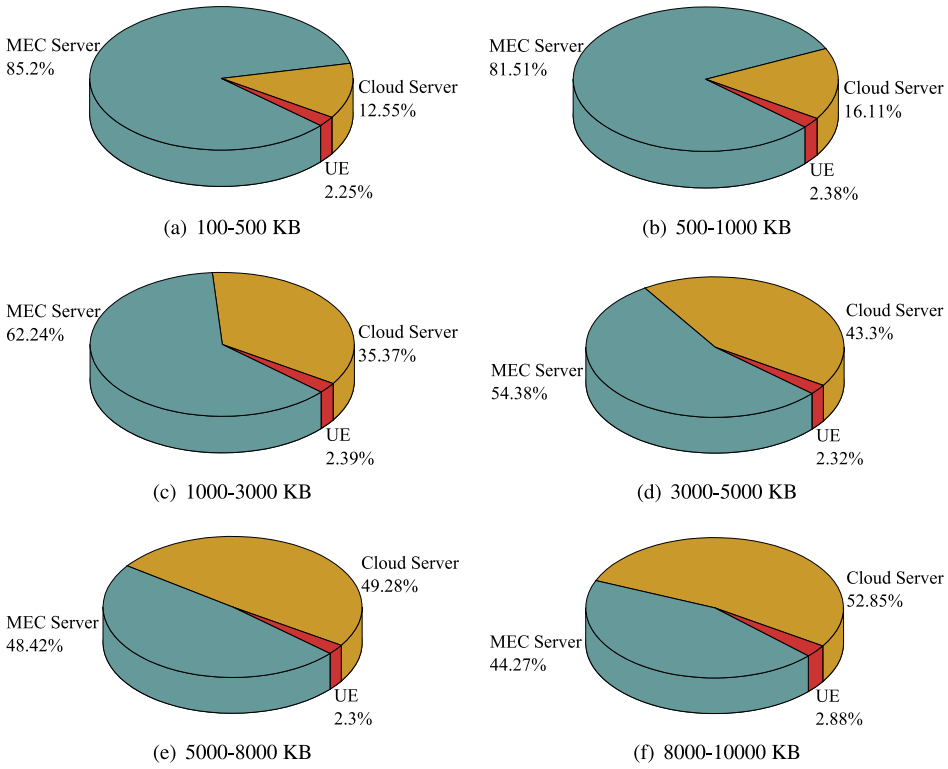


Fig. 4. Proportion of tasks executed on different computing layers.

receive the computational load decreases, while the cloud server has a stronger load capacity. This is because the computing capabilities and resources of different computing layers are different, and the computing capabilities and resources of the UE, MEC server, and cloud server are increased by multiples. When the calculation pressure in the environment increases, more tasks will be offloaded to the cloud because the computing capabilities and resources of the UE devices and edge layer are insufficient for supporting an excessive computational load. However, when the calculation pressure is small, the edge computing layer's unique advantage of providing services close to the user can best satisfy the user's QoE. This experiment shows that the cooperation between the edge computing layer and cloud computing layer can effectively complete the processing of user requests.

5.2. Task data size

Another experiment in this paper simulate the task generation and processing procedures for 5 h, and the experimental results are recorded at time points of 0.5, 1.0, 3.0, and 5.0 h. This experiment compares the average response times and total energy consumption results of various algorithms under different task data sizes and summarizes the tasks into small tasks (100–500 KB), large tasks (1000–3000 KB) and mixed tasks (100–3000 KB) according to the data size. The experimental results are shown in Figs. 5 and 6.

Figs. 5(a) to 5(c) show the average response time results of the small task, mixed task, and large task, respectively. As the task data size increases, the task response time also increases. Since the mixed task includes small tasks and large tasks, the values of its experimental results are between the experimental results of small tasks and large tasks. The experimental results of the SLTRA algorithm are optimal, and those of the QL algorithm are second. This is because SLTRA and QL are intelligent algorithms based on self-learning methods, and through interaction with the experimental environment, they can learn better task offloading and resource allocation strategies than other approaches. As the learning time increases, an algorithm's decision accuracy can be improved, so the objective optimization performance is also improved. However, because the SLTRA algorithm is based on the DRL method and has better learning ability and stability than the QL algorithm, the optimization performance of the QL algorithm is weaker than that of the SLTRA algorithm. Next, the

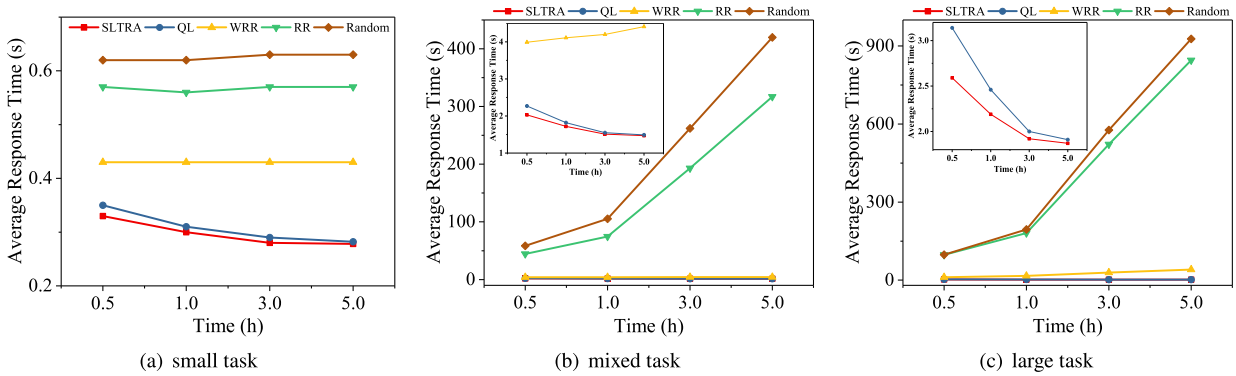


Fig. 5. Average response times for different task data sizes.

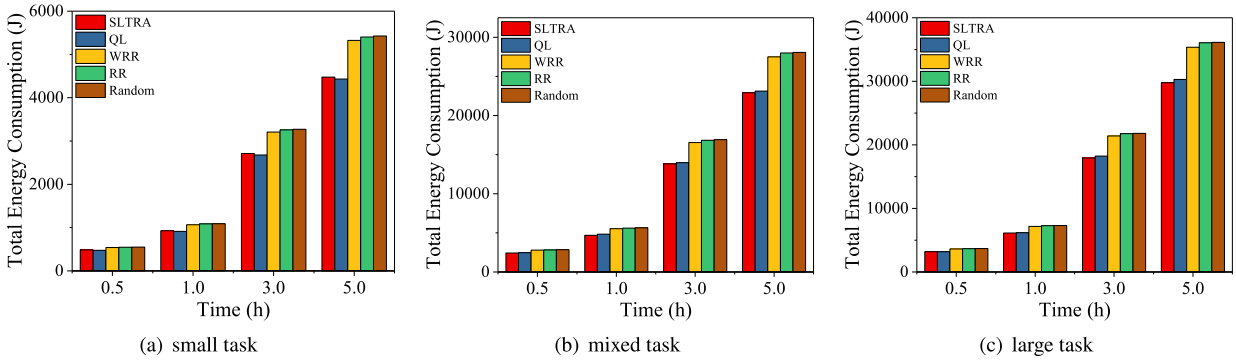


Fig. 6. Total energy consumption for different task data sizes.

optimization performances of the WRR, RR, and random algorithms decline in the listed order. These three basic algorithms do not have the ability to learn strategies, so their performance advantages are limited.

Figs. 6(a) to 6(c) show the total energy consumption of the small task, mixed task, and large task, respectively. Similar to the experimental results with respect to the task response time, the total energy consumption increases with the task size increasing. With the increasing task size, the number of tasks generated by UE devices increases, and the energy consumption required for processing tasks also increases. According to the experimental results, under the small task size, the total energy consumption of the SLTRA algorithm is slightly higher than that of the QL algorithm. However, for large and mixed tasks, the performance of the SLTRA algorithm is better than those of the other comparison algorithms. Among the remaining three algorithms, the experimental results of the WRR algorithm are relatively good, but the improvement of the optimization effect is not obvious, and all of these basic algorithms are worse than SLTRA and QL algorithms.

According to the experimental results regarding the average response times and total energy consumption for the three types of tasks, it can be seen that the task offloading and resource allocation algorithm proposed in this paper has the best effects, which reflects the good optimization performance and the algorithm robustness. This is because the SLTRA algorithm considers both the average task response time and the total energy consumption.

5.3. Number of UE devices

The number of mobile UE devices directly affects the number of user requests. The greater the number of UE devices is, the greater the pressure on the computing environment. In this experiment, we optimize the objectives of minimizing the average response times and total energy consumption of tasks under different numbers of UE devices, including 5, 10, 15, and 20

Figs. 7(a) to 7(d) are the results regarding the average task response times under different numbers of UE devices. First, with the increase in the number of UE devices, the time cost required to process all tasks increases correspondingly. However, the number of tasks generated by the UE devices also increases, so the average task response time does not change significantly. Second, with the increase in UE activity time, the number of tasks generated by the UE devices increases. This has no obvious impact on the experimental results of the SLTRA and QL algorithms, and the performances of the algorithms improve as the training time increases due to the adaptive learning ability of these approaches. However, the average task response times of the WRR, RR, and random algorithms increase as the activity time increases, especially for the RR and random algorithms. This is because the RR and random algorithms do not consider the computing capability differences between nodes and do not balance the computing pressure among the computing nodes, resulting in too many tasks accumulating on computing nodes with weak capability, and the average response time increases as a result. According to the comparison results of different algorithms, the SLTRA algorithm has the best optimization results, and the QL algorithm is second only to the SLTRA algorithm due to its learning ability.

Figs. 8(a) to 8(d) show the results of the total energy consumption for task processing under different numbers of UE devices.

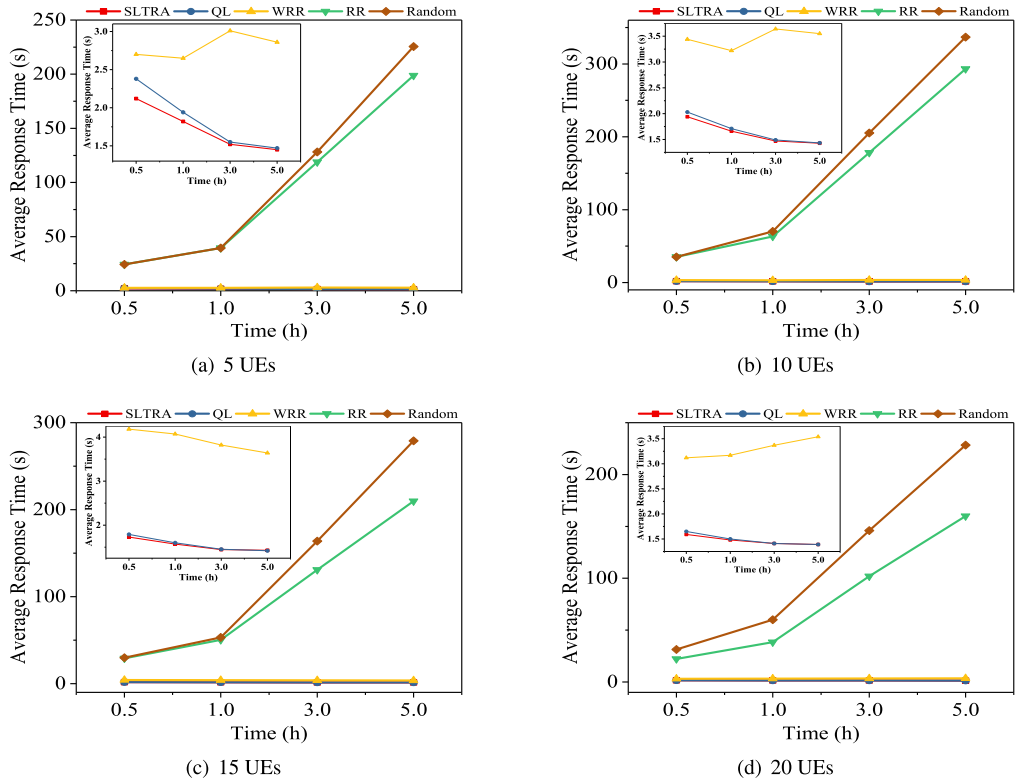


Fig. 7. Average task response times for different numbers of UE devices.

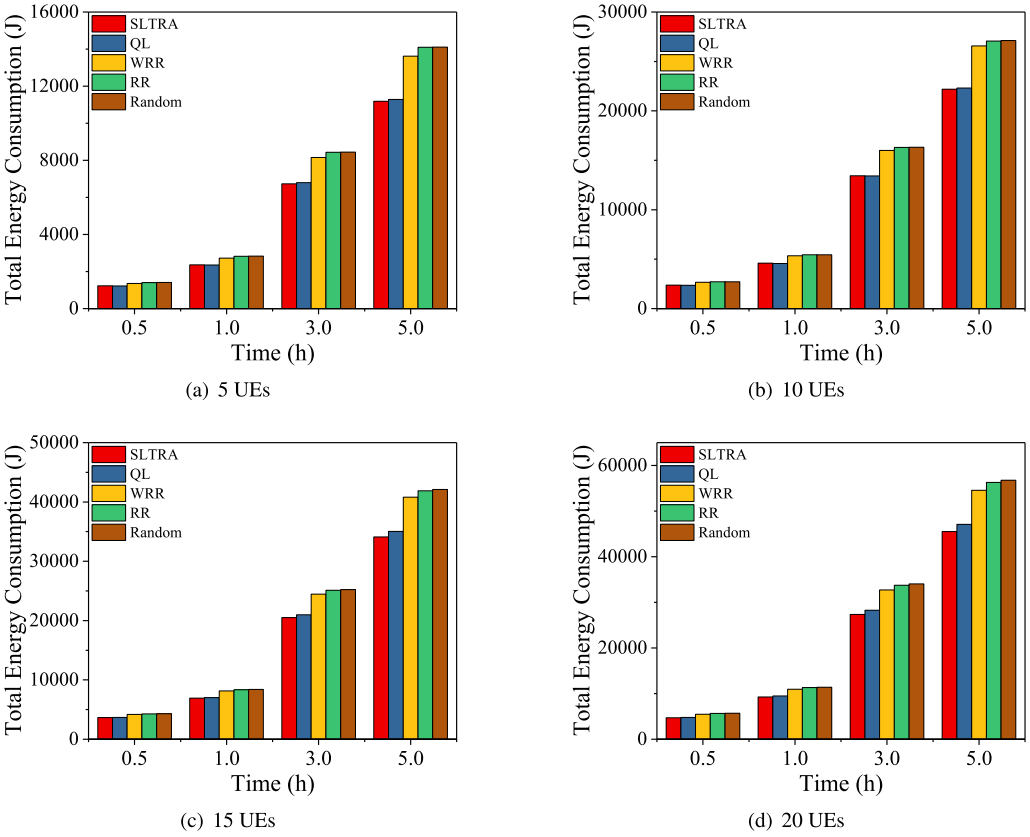


Fig. 8. Total energy consumption for different numbers of UE devices.

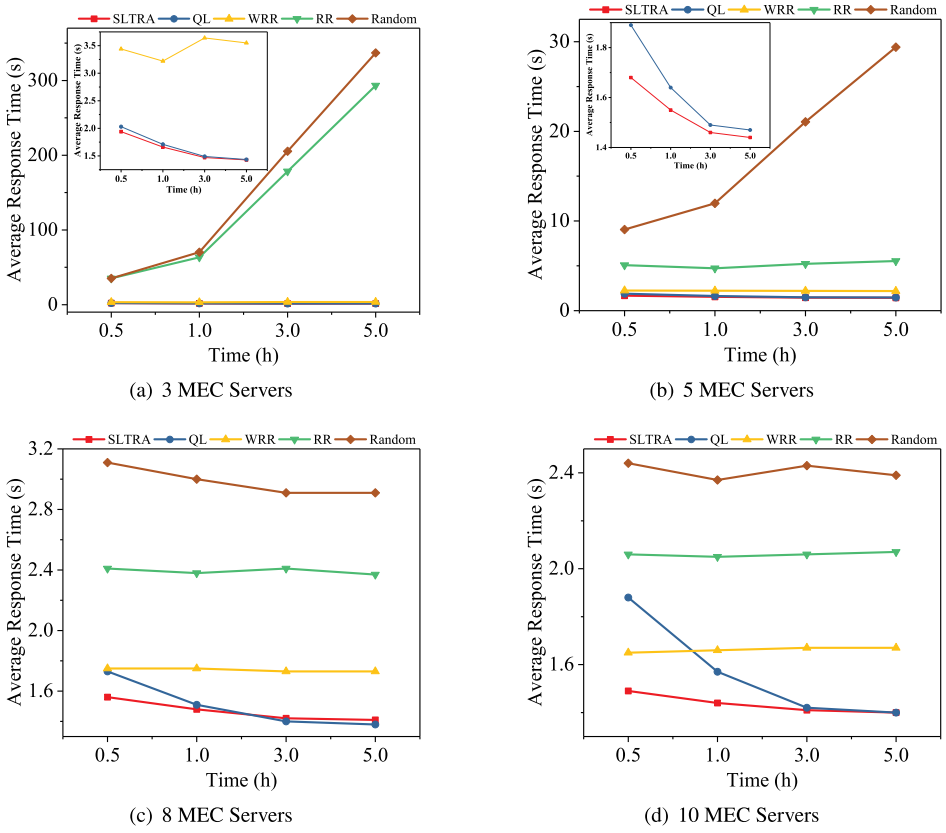


Fig. 9. Average tasks response time for different numbers of MEC servers.

First, when the number of UE devices or the device activity time increases, the number of tasks to be processed also increases, so the total energy consumption required to complete all tasks increases as well. Second, the experimental results of the SLTRA and QL algorithms are better than those of the other three comparison algorithms, and the SLTRA algorithm is better than the QL algorithm. The WRR, RR, and random algorithms have similar results in terms of energy consumption optimization.

In conclusion, the SLTRA algorithm is the best at optimizing both the average response time and the total energy consumption for different numbers of UE devices. This is because the SLTRA algorithm is based on adaptive learning for making decisions about a given optimization problem, and the optimization performance of the algorithm is correspondingly improved as the learning time increases.

5.4. Number of MEC servers

For task processing, in addition to the computing capabilities and resources of the computing nodes, the number of computing nodes is also an important influencing factor. Since the computing node of the local computing model is a mobile UE and the computing node of the cloud computing model is a cloud server with sufficient capability and resources, this experiment is performed for different numbers of MEC servers. The number of MEC servers is set to four separate values, respectively 3, 5, 8, and 10. The average response times and total energy consumption for tasks under different numbers of MEC servers are shown in Figs. 9 and 10, respectively.

Figs. 9(a) to 9(d) are the results regarding the average task response times under different numbers of MEC servers. First, it is clear that the average response time and the MEC servers' numbers are negatively correlated. The reason is that growing MEC servers leads to increased computing capability and resources, so the nodes can execute more tasks in parallel, reduce the waiting times of tasks, and thus reduce the task response time. Second, with increasing UE activity time, the number of tasks also increases. In the case with a small number of MEC servers, the task response time increases significantly as the activity time increases. However, with the increase of MEC servers, the task response time increases slowly as the activity time decreases and even decreases at some points. This shows when MEC servers is not enough, the computing capability and resources of the computing platform are insufficient for processing many tasks. Therefore, when the activity time increases, the task response time increases significantly. The increase in the number of MEC servers increases the calculation capability and resources, and so there are sufficient resources on the computing platform to perform tasks; thus, the task response time is not affected. From the comparison between the algorithms, the experimental results of the SLTRA algorithm are optimal in most cases and slightly worse than those of the QL algorithm in a few cases, depending on the decision made by the SLTRA algorithm to balance the optimization of the task response time and energy consumption. The optimization performances of the WRR, RR, and random algorithms decrease in the listed order. With the increase in the number of MEC servers, the optimization performance of the RR algorithm is improved.

Figs. 10(a) to 10(d) are the total energy consumption results with 3, 5, 8 and 10 MEC servers, respectively. It is clear that the increasing servers does not effects energy consumption a lot because the total energy consumption mainly depends on the number of tasks and does not depend on the number of

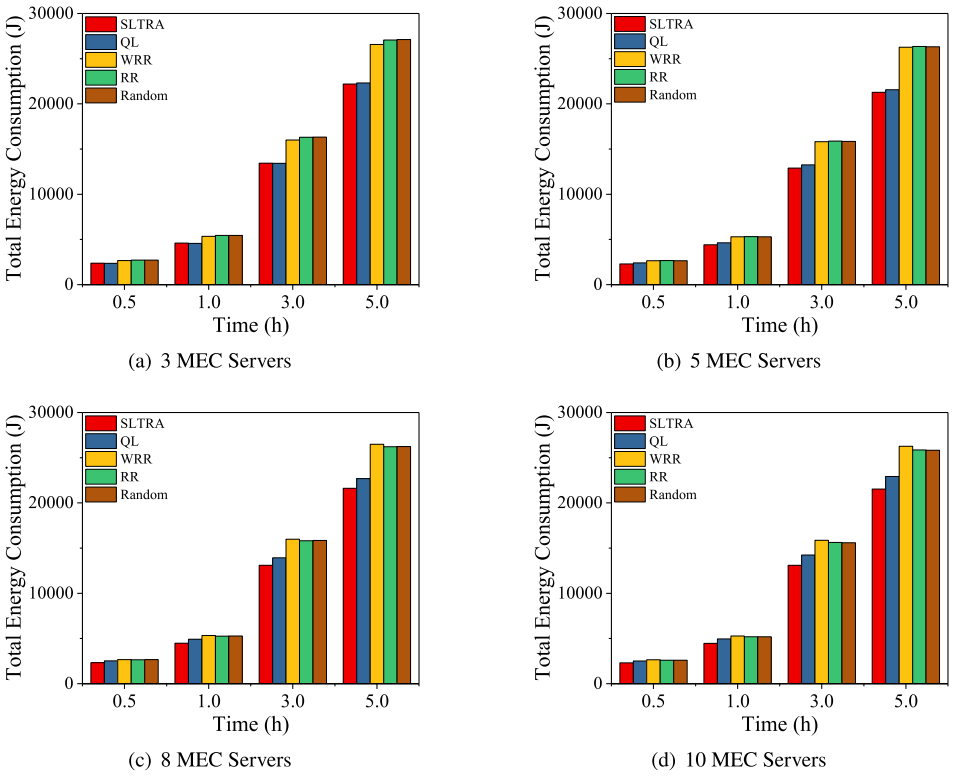


Fig. 10. Total energy consumption for different numbers of MEC servers.

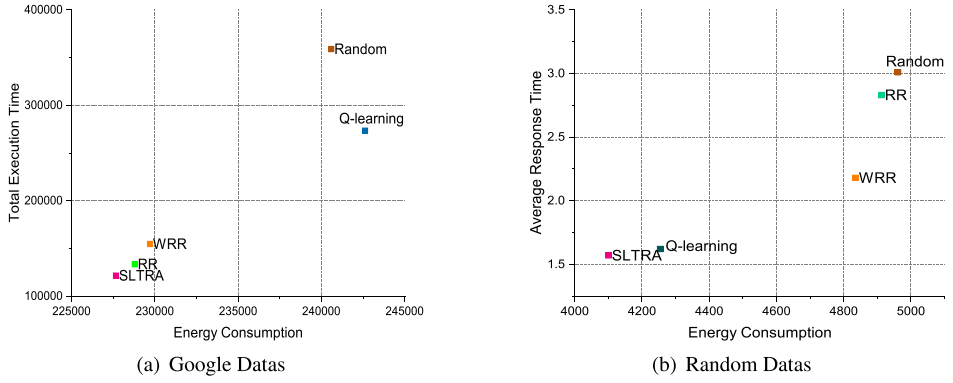


Fig. 11. Pareto scatter plot for different data sets.

computing nodes. Therefore, as the UE activity time increases, the task energy consumption also increases. Next, we compare all the comparison algorithms basing the optimization goals. The performance of the SLTRA algorithm is still the best, followed by that of the QL algorithm. The experimental results of the WRR, RR, and random algorithms with respect to the total task energy consumption are similar.

Therefore, the average response time and total energy consumption results are combined under different numbers of MEC servers. The SLTRA algorithm is better than other comparison algorithms due to its good ability to learn an effective strategy, and the QL algorithm, which also has a certain learning ability, is second. The performances of the WRR, RR, and random algorithms are relatively poor and decrease in turn.

5.5. Pareto scatter

In the above experiments, we have tested the optimization effects of these five algorithms on average response time and total energy consumption under different conditions. However, this is not enough to clearly reflect the ability of the algorithm to optimize multiple targets at the same time. Thus, the Pareto scatter plot experiment is conducted to prove the algorithms' joint optimization ability to make up for this flaw.

Figs. 11(a) is the Pareto scatter plots of different algorithms for total energy consumption and total execution time using Google cluster datas. Figs. 11(b) is the Pareto scatter plots of random datas for total energy consumption and Average Response time. It can be seen our proposed algorithm performs better among these five algorithms in both figures. The SLTRA algorithm achieved the minimum in these two objective targets at the same time.

Conclusion and future work

In this paper, we propose an adaptive algorithm under the constraints of system resources and reliability based on the DRL method, and the proposed approach effectively solves the problems of collaborative management and optimization in task offloading and resource allocation. The Poisson distribution is used to simulate the process of UE task generation, and the mobility of UE devices is considered. In addition to being processed locally, tasks can be offloaded to edge servers or cloud servers for processing. Simulation results show that the SLTRA algorithm has the best performance in terms of objective optimization, improves users' QoE, and saves system energy costs.

In future work, we will further consider realistic factors, make the model closer to the real situation. Such as the instability of network transmission, tasks need to communicate collaboratively, and the standby energy consumption of computing nodes in the computing model, etc. At the same time, real data sets and experimental platforms need to be used to test experimental results.

CRediT authorship contribution statement

Zhao Tong: Idea, Survey, Optimization, Reviewing, Writing. **Xiaomei Deng:** Survey, Implement, Experiments, Writing – original draft, Writing – review & editing. **Jing Mei:** Optimization, Reviewing. **Bilan Liu:** Polishing the paper, Modifying the grammar. **Keqin Li:** Suggestions, Reviewing.

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.

Acknowledgments

This work was supported by the Program of National Natural Science Foundation of China (grant No. 62072174, 61602170, 61872129) and the National Natural Science Foundation of Hunan Province, China (grant No. 2020JJ5370).

References

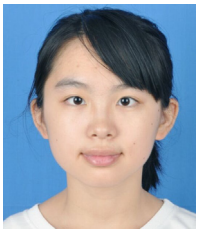
[1] C. Cicconetti, M. Conti, A. Passarella, A decentralized framework for serverless edge computing in the internet of things, IEEE Trans. Netw. Serv. Manag. PP (99) (2020) 1.
[2] M. Abbasi, E. Mohammadi-Pasand, M.R. Khosravi, Intelligent workload allocation in IoT-fog-cloud architecture towards mobile edge computing, Comput. Commun. (2021).
[3] Z. Wang, Y. Zhuang, Z. Yan, TZ-MRAS: A remote attestation scheme for the mobile terminal based on ARM TrustZone, Secur. Commun. Netw. 2020 (7) (2020) 1–16.
[4] H. Ning, Y. Li, F. Shi, L.T. Yang, Heterogeneous edge computing open platforms and tools for internet of things, Future Gener. Comput. Syst. 106 (2020) 67–76.
[5] S. Bi, L. Huang, Y.J.A. Zhang, Joint optimization of service caching placement and computation offloading in mobile edge computing systems, IEEE Trans. Wireless Commun. PP (99) (2020) 1.
[6] F. Wang, J. Xu, S. Cui, Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems, IEEE Trans. Wireless Commun. 19 (4) (2020) 2443–2459.
[7] S. Kim, New application task offloading algorithms for edge, fog, and cloud computing paradigms, Wirel. Commun. Mob. Comput. 2020 (2020) 1–14.
[8] K. Li, A game theoretic approach to computation offloading strategy optimization for non-cooperative users in mobile edge computing, IEEE Trans. Sustain. Comput. (2018) http://dx.doi.org/10.1109/TSUSC.2018.2868655.
[9] Q. Zhang, Q. Zhang, W. Shi, H. Zhong, Firework: Data processing and sharing for hybrid cloud-edge analytics, IEEE Trans. Parallel Distrib. Syst. 29 (9) (2018) 2004–2017.
[10] F. Wang, B. Diao, T. Sun, Y. Xu, Data security and privacy challenges of computing offloading in FINs, IEEE Netw. 34 (2) (2020) 14–20.
[11] T. Liu, L. Fang, Y. Zhu, W. Tong, Y. Yang, Latency-minimized and energy-efficient online task offloading for mobile edge computing with stochastic heterogeneous tasks, IEEE Trans. Mob. Comput. PP (99) (2020) 1.
[12] J. Feng, F.R. Yu, Q. Pei, X. Chu, J. Du, L. Zhu, Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach, IEEE Internet Things J. 7 (7) (2020) 6214–6228.
[13] T. Chen, S. Barbarossa, X. Wang, G.B. Giannakis, Z.L. Zhang, Learning and management for internet of things: Accounting for adaptivity and scalability, Proc. IEEE (2019).
[14] H. Huang, K. Peng, X. Xu, Collaborative computation offloading for smart cities in mobile edge computing, in: 2020 IEEE 13th International Conference on Cloud Computing, CLOUD, 2020.
[15] Z. Tong, X. Deng, F. Ye, S. Basodi, Y. Pan, Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment, Inform. Sci. 537 (2020).
[16] M.-H. Chen, M. Dong, B. Liang, Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints, IEEE Trans. Mob. Comput. 17 (12) (2018) 2868–2881.
[17] L. Liu, Z. Chang, X. Guo, S. Mao, T. Ristaniemi, Multiobjective optimization for computation offloading in fog computing, IEEE Internet Things J. 5 (1) (2017) 283–294.
[18] K. Kofler, I. Grasso, B. Cosenza, T. Fahringer, An automatic input-sensitive approach for heterogeneous task partitioning, in: ICS 2013, 2013.
[19] J. Du, L. Zhao, J. Feng, X. Chu, Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee, IEEE Trans. Commun. 66 (4) (2018) 1594–1608.
[20] M. Mukherjee, S. Kumar, M. Shojafar, Q. Zhang, C.X. Mavromoustakis, Joint task offloading and resource allocation for delay-sensitive fog networks, in: ICC 2019-2019 IEEE International Conference on Communications, ICC, IEEE, 2019, pp. 1–7.
[21] D. Grewe, M.F.P. O'Boyle, A static task partitioning approach for heterogeneous systems using opencl, in: J. Knoop (Ed.), Compiler Construction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 286–305.
[22] Y. Hao, Y. Jiang, T. Chen, D. Cao, M. Chen, itaskoffloading: Intelligent task offloading for a cloud-edge collaborative system, IEEE Netw. 33 (5) (2019) 82–88.
[23] Z. Tong, X. Deng, H. Chen, J. Mei, DDMTS: A novel dynamic load balancing scheduling scheme under sla constraints in cloud computing, J. Parallel Distrib. Comput. 149 (2020).
[24] J. Hu, C. Liu, K. Li, K. Li, Game-based multi-MD with QoS computation offloading for mobile edge computing of limited computation capacity, in: IFIP International Conference on Network and Parallel Computing, Springer, 2019, pp. 16–27.
[25] Y. Ouyang, W. Liu, Q. Yang, X. Mao, F. Li, Trust based task offloading scheme in UAV-enhanced edge computing network, Peer-To-Peer Netw. Appl. (4) (2021) 1–23.
[26] W. Wu, F. Zhou, R.Q. Hu, B. Wang, Energy-efficient resource allocation for secure NOMA-enabled mobile edge computing networks, IEEE Trans. Commun. 68 (1) (2020) 493–505.
[27] J. Yan, S. Bi, Y.J.A. Zhang, Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach, IEEE Trans. Wireless Commun. 19 (8) (2020) 5404–5419.
[28] S.S. Lee, S.K. Lee, Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information, IEEE Internet Things J. PP (99) (2020) 1.
[29] C. Huang, G. Huang, W. Liu, R. Wang, M. Xie, A parallel joint optimized relay selection protocol for wake-up radio enabled WSNs, Phys. Commun. 47 (2021) 101320, http://dx.doi.org/10.1016/j.phycom.2021.101320, URL https://www.sciencedirect.com/science/article/pii/S1874490721000574.
[30] Q. Zhang, L.T. Yang, Z. Yan, Z. Chen, P. Li, An efficient deep learning model to predict cloud workload for industry informatics, IEEE Trans. Ind. Inf. 14 (7) (2018) 3170–3178.
[31] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep Q-learning, Inform. Sci. 512 (2020) 1170–1191.
[32] C. Sonmez, A. Ozgovde, C. Ersoy, Edgecloudsim: An environment for performance evaluation of edge computing systems, Trans. Emerg. Telecommun. Technol. 29 (11) (2018) e3493.
[33] X. Tao, K. Ota, M. Dong, H. Qi, K. Li, Performance guaranteed computation offloading for mobile-edge cloud computing, IEEE Wirel. Commun. Lett. 6 (6) (2017) 774–777.
[34] Y. Mao, J. Zhang, K.B. Letaief, Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems, in: 2017 IEEE Wireless Communications and Networking Conference, WCNC, IEEE, 2017, pp. 1–6.

[35] X. Qu, Y.-S. Ong, Y. Hou, X. Shen, Memetic evolution strategy for reinforcement learning, in: 2019 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2019, pp. 1922–1928.
[36] Z. Tong, X. Deng, H. Chen, J. Mei, H. Liu, QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment, Neural Comput. Appl. (2019) 1–18.
[37] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[38] X. Han, H. Liu, F. Sun, X. Zhang, Active object detection with multistep action prediction using deep Q-network, IEEE Trans. Ind. Inf. 15 (6) (2019) 3723–3731.
[39] A. Jindal, A. Dua, K. Kaur, M. Singh, N. Kumar, S. Mishra, Decision tree and SVM-based data analytics for theft detection in smart grid, IEEE Trans. Ind. Inf. 12 (3) (2016) 1005–1016.

Zhao Tong received his Ph.D. in computer science from Hunan University, China, in 2014. He was a visiting scholar with Georgia State University from 2017 to 2018. He is currently an Associate Professor at the College of Information Sciences and Engineering of Hunan Normal University. His research interests include modeling and scheduling for parallel and distributed computing systems. He has published more than 20 research papers in international conferences and journals, such as IEEE Transactions on Parallel and Distributed Systems, Information Sciences, Journal of Parallel and Distributed Computing, etc. He is a member of the IEEE and CCF.

Xiaomei Deng is currently pursuing a master's degree at the College of Information Sciences and Engineering of Hunan Normal University. Her research interests include scheduling for cloud computing, edge computing, artificial intelligence, machine learning and combinatorial optimization. She has published 4 research articles in international conferences and journals.

Jing Mei received her Ph.D. in computer science from Hunan University, China, in 2015. She is currently an assistant professor at the College of Information Sciences and Engineering of Hunan Normal University. Her research interests include parallel and distributed computing and cloud computing. She has published more than 12 research articles in international conferences and journals, such as IEEE Transactions on Computers, IEEE Transactions on Service Computing, Cluster Computing, the journal of Grid Computing, and the Journal of Supercomputing.



Bilan Liu is pursuing a master's degree at the College of Information Sciences and Engineering of Hunan Normal University. Her research interests include Cloudedge collaborative computing, deep learning algorithms and combinatorial optimization.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has published over 680 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.