Journal of Parallel and Distributed Computing 169 (2022) 185–198

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

A novel task offloading algorithm based on an integrated trust mechanism in mobile edge computing

Zhao Tong [a,*], Feng Ye [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*

A R T I C L E   I N F O

Article history:
Received 7 July 2021
Received in revised form 15 May 2022
Accepted 7 July 2022
Available online 19 July 2022

Keywords:
Deep reinforcement learning
Integrated trust mechanism
Mobile edge computing
Task offloading

A B S T R A C T

As a new computing model, mobile edge computing (MEC) is designed to better deal with various forms of service requests, such as computing intensity and delay sensitivity, in the era of big data and the Internet of Things (IoT). However, the development of MEC is still in its infancy, and many issues need to be further investigated. One of the key issues that needs to be addressed in MEC is rational task offloading. Due to the dynamic, real time and complex nature of the MEC environment, the security and reliability of edge data are becoming increasingly important. Based on the above problems, we construct a task offloading integrated trust evaluation mechanism and, combined with the double deep Q-network (DDQN) algorithm in deep reinforcement learning (DRL), propose a novel task offloading algorithm, named DDTMOA. Simulation results show that the DDTMOA algorithm can effectively reduce the average task response time and total system energy consumption while ensuring task offloading performance compared to other classical algorithms.

1. Introduction

With the advent of the 5G network era, 5G communication technology is gradually moving toward standardization. The IoT [2] and in-vehicle networks (IVNs) [17] are increasingly widely used in people's daily lives. With the increase in terminal equipment, data present explosive growth. According to the latest results of the Cisco global cloud index, in 2021, the quantity of data generated by various terminal devices, humans, and machines reached 3.2 zettabytes (ZB) [11]; additionally, Huawei predicts that by 2025, the number of terminal devices connected to the Internet of Things will be infinitely close to 100 billion [8].

MEC [1], as an effective supplement to cloud computing, gathers various resources, such as computing, storage, intelligence, and application, at the edge of the network (near the user end). Part of the user's service requests can be offloaded to a near-end MEC server for processing through the wireless channel so that delay-sensitive tasks can respond quickly, reduce task response time, and improve the user's Quality of Service (QoS) and Quality of Experience (QoE). Some tasks are offloaded to the MEC server, which can alleviate network bandwidth transmission pressure and reduce the transmission energy consumption and transmission overhead. Some of the computationally intensive tasks can be offloaded to a remote cloud center with more sufficient computing and storage resources via edge base stations (BSs) through the core network for processing so that computationally intensive service requests can also be guaranteed. Therefore, reasonable task offloading (computing offloading) is particularly important. Because of real-time dynamics and edge environment complexity, edge environment security, such as data offloading security, is also facing a great challenge. The reliability and security of edge data are becoming increasingly prominent [21], [9]. Therefore, to ensure the reliability and security of edge data, new methods of offloading data security tasks for edge computing are urgently needed.

Design efficient task scheduling and resource allocation strategies is one of the ways to enhance the task offloading experience of mobile users in the MEC distributed systems. Currently, an increasing number of scholars are using reinforcement learning to solve this problem [13], [34]. Among them, the Q-learning algorithm determines the value of each state corresponding to the next action by constructing a Q-table. However, the low-dimensional inputs and outputs are not able to satisfy complex MEC scenarios. In the next step, many complex strategies use DRL to allocate tasks as well as resources [37]. In conjunction with deep learning, neural networks are used instead of the Q-table. While this approach enables deep Q network (DQN) to handle high-dimensional data, it tends to be limited to falling into suboptimal solutions [19], [25]. Therefore, to obtain a better offloading strategy when the num-

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

https://doi.org/10.1016/j.jpdc.2022.07.006
0743-7315/© 2022 Elsevier Inc. All rights reserved.

ber of end devices is quite large, we design an offloading strategy based on the DRL algorithm DDQN. However, simply establishing such an offloading strategy is not enough. Because there will be an other significant problem-trust management that should be considered [22], [30]. During the offloading process, the user's identity authority and the trustworthiness of the compute node may pose a threat to security and privacy. To reduce these security risks, we propose a trust mechanism considering three aspects: identity, behavior, and capability.

Therefore, this paper studies the task offloading problem based on the integrated trust evaluation mechanism combined with DDQN [16] and proposes a novel task offloading algorithm, which provides a new idea for solving the task offloading problem in the MEC environment and has certain value and significance for future theoretical research and practical application significance.

The main contributions of this paper are as follows:

- To ensure the reliability, privacy, and security of edge data when solving the task offloading problem in mobile edge computing, we propose a task offloading framework based on an integrated trust evaluation mechanism. The mechanism consists of trusted identity, trusted behavior, and trusted capability.
- We build a two-tier MEC model, and based on Markov decision theory, the task offloading problem under the two-tier MEC model is modeled as a Markov decision process (MDP), and explore the optimal strategy through the DRL method. We integrate the trust mechanism into the constraints of the model, which is a prerequisite for performing DRL.
- We study a biobjective problem in a two-tier MEC environment. To objectively describe the proportion of two optimization objectives in the MEC scene, we use the entropy weight method to estimate the weight value of the optimization objectives.
- Based on the integrated trust mechanism and combined with the DDQN algorithm, we propose a novel task offloading algorithm. Experimental results show that our proposed algorithm has better performance than other algorithms.

The rest of this paper is organized as follows. The Section 2 reviews related work on the task offloading problem in the MEC environment. The Section 3 describes the MEC model, task type and definition, communication model, computation model, integrated trust mechanism, and task offloading problem. The Section 5 explains the MDP model, the DRL method and DDQN algorithm, and the main content and design principle of our algorithm. The Section 6 compares the performance of our proposed algorithm with that of several other algorithms and analyzes the experimental results. Finally, the Section 7 summarizes this paper and provides potential directions for future work.

2. Related work

MEC, as a new computing model, has been widely considered by academia and industry since it was proposed. Many scholars have carried out in-depth research on MEC and have fully considered the security, reliability and privacy of edge data, task offloading and resource allocation in the MEC environment. Efficient task offloading maximizes the benefits of user providers and improves the QoS and QoE of users as much as possible under limited resources. Task offloading is a typical NP-hard problem [10].

At present, for the task offloading problem in an MEC environment, the main optimization objectives include energy consumption, delay, and cost. For different optimization objectives, many optimization methods have been proposed. To minimize system energy consumption, the profits of mobile service providers (MSPs) are maximized. Wang et al. [28] studied a unified MSP performance tradeoff framework, using Lyapunov technology to optimize the framework, and the VariedLen algorithm to solve the optimization problem. The experimental results show that this method can make the average MSP profit reach the optimal level under the premise of ensuring system stability and low congestion. To improve the energy efficiency of an MEC system, Wang et al. [31] proposed the joint energy minimization and resource allocation problem of CRAN and MEC and transformed the problem into a nonconvex optimization problem. Under the delay constraint, an iterative algorithm was used to solve the optimization problem to minimize the weighted sum of the two kinds of energy. The simulation results show that the method can improve system performance and save energy.

The above task offloading optimization methods can improve the performance of MEC systems to a certain extent. However, the objectives of the above methods are relatively singular, and there may be a case of optimizing another objective at the cost of one resource. Therefore, the optimization of a single objective is not enough to verify that an optimization method is absolutely effective. [35] designed a task offloading optimization framework. Under this framework, an optimization problem was proposed to minimize both task execution delay and offloading failure probability. The problem was transformed into a nonconvex optimization problem and solved by a heuristic algorithm. Numerical simulation results show that the proposed method can trade off task delay and energy consumption with less complexity.

The above methods consider the biobjective and multiobjective problems in MEC and solve the optimization problems through linear optimization, nonlinear optimization, convex optimization, combinatorial optimization and other mathematical methods, which reflects MEC performance to a certain extent. However, the above pure mathematical combination optimization methods have great limitations, the solution process is complex, the time and space complexity is high, and the universality is not strong, which is more reflected in the value of theoretical research. With the great breakthrough in artificial intelligence (AI), DRL has made considerable breakthroughs in the field of AI. In recent years, the DRL method has also been widely used in cloud computing and MEC scenarios. To solve the problem of multiservice node offloading and mobile task multidependence in large-scale heterogeneous MEC, Lu et al. [18] developed a new task offloading algorithm based on DRL and improved the algorithm by using an LSTM network. Simulation results showed that the algorithm is superior to other algorithms in energy consumption, load balancing, delay and average execution time.

As mentioned above, the task offloading and resource allocation algorithm based on DRL can improve the performance of the MEC system to a certain extent. However, MEC wants to achieve comprehensive development, and the security, privacy and reliability of edge data must be guaranteed. Elgendy et al. [7] created a multi-user resource allocation and computational offloading model with data security, taking into account the computational capacity, resource constraints and data security of the MEC system. The model introduces AES encryption as a security layer to protect sensitive information from network attacks. Finally, the least squares method is used to solve the task offloading problem. Simulation results show that the method can effectively improve the performance of the system compared to local execution and complete offloading schemes. To ensure data security and integrity, Hou et al. [5] combined blockchain technology with MEC to propose a task allocation problem that considers node capacity and reward fairness, and solved it by a heuristic algorithm. To ensure the credibility of the computational results, a police patrol model was used to optimize the overall rewards of the system. Experimental results demonstrate the effectiveness of the algorithm.

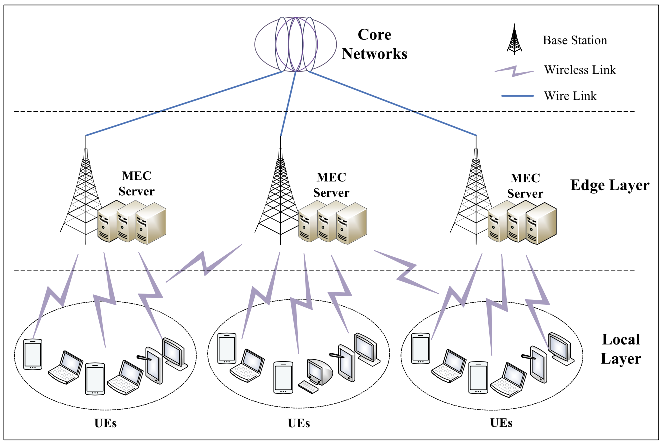

Fig. 1. Two-tier MEC model.

Currently, although researchers have carried out some research on related issues in MEC, most of the related research is in the initial stage, and many technologies are not mature enough. The traditional solutions to the security, reliability and privacy issues of cloud computing are not fully applicable to MEC. Therefore, optimizing the performance of MEC on the premise of ensuring the security and reliability of edge data is a problem worthy of further study. For this reason, this paper proposes an integrated trust evaluation mechanism. Based on this mechanism, consider optimizing dual objectives, including delay and energy consumption, combined with the DDQN [25], [32] algorithm is used to solve the task offloading problem in MEC. Experimental results show that our proposed algorithm can effectively reduce the average task response time and the total system energy consumption while ensuring task offloading performance.

3. Overview of MEC model architecture

In this section, we describe the MEC model first. After that, we introduce the task type and definition. And we introduce the system submodels, including the communication model and the computation model.

3.1. MEC model

As shown in Fig. 1, in this paper, we consider a two-layer MEC model; the two layers are the local terminal equipment layer and the edge server layer. The local terminal equipment layer is composed of various user equipment (UE) devices, and the local equipment has certain computing and storage capabilities.

The edge server layer is composed of base stations (BSs) and edge servers. Edge servers have more computing and storage resources and stronger task processing abilities than local devices. Each base station (BS) and multiple heterogeneous servers form an edge area, and the coverage of the edge area is limited. The UE communicates with the BS through the wireless network, and the BS interacts through the core network. The communication distance between UEs and BSs is different.

Task offloading can be divided into two types: partial offloading and overall offloading. Partial offloading refers to decomposing an independent task into multiple subtasks; some subtasks are offloaded from the UE to the edge server for execution, and the remaining partial subtasks are reserved for processing on the local UE. Overall offloading refers to offloading all tasks to the edge server for processing. This paper considers the overall offloading of tasks in the MEC model, and the tasks can only be processed on the edge server or UE alone. In the MEC model we consider, it is assumed that the task needs to be offloaded from the local end to the edge server for processing. According to the task offloading strategy, in general, the task will be offloaded to the edge server near the edge area for execution. However, the task may also be offloaded to other edge servers in the far edge area, which will lengthen the communication distance and increase the task communication delay.

3.2. Task type and definition

The big data era presents a variety of service requests (task requests), which can be divided into different categories according to different task attributes. Considering whether there are dependencies between tasks, tasks can be divided into independent tasks and nonindependent tasks. Independent tasks cannot be further subdivided, and there is no constraint relationship between tasks; nonindependent tasks include multiple tasks, and there are priority and constraint relationships between subtasks. Subtasks must wait for the completion of all parent tasks before they can start execution. Considering whether the task request is real time, the tasks can be divided into online tasks and offline tasks. Offline tasks indicate that the broker already knows the total number of tasks arriving at the data center, the arrival time of each task, the quantity of data, the requested computing resources and the storage resources before starting to schedule and unload tasks. Online tasks refer to tasks that are generated in real time. Before scheduling and unloading tasks, the broker does not know how many tasks will arrive in the data center and all related attributes of the tasks. Each task is processed every time it arrives. MEC connects the wireless network and AI together, generally including the cloud, edge and user terminals. The user terminal generally refers to a mobile device such as mobile phones and notebook computers, and the tasks generated by these devices are small and independent. Therefore, this article simulates online independent tasks, the task queue is nonpreemptive, and the time interval for task generation to arrive at the data center obeys a Poisson distribution. Task attributes can be defined as:

$$task = \{id_i, sub_i, d_i, mem_i, cpu_i, deadline_i, sct_i, area_i\}, \tag{1}$$

where id_i is the id of task i, sub_i is the submission time of task i, d_i represents the data size of task i, mem_i denotes the memory resources requested by task i, cpu_i indicates the computing resources requested by task i, $deadline_i$ represents the maximum response time that task i can tolerate, sct_i denotes the minimum security level requirement for the computing node when task i requests execution, and $area_i$ indicates the generation area of task i.

3.3. Communication model

The UE communicates with the edge BS through the wireless network. Assuming that the channel gain between the UE and the edge BS is g, the unit is dB, and the channel gain can be expressed as:

$$g = 127 + 25 \cdot \log 10\,(D), \tag{2}$$

where D represents the communication distance between the UE and the edge BS. Assuming that task i needs to be offloaded to an edge server for execution, the transmission power of task i offloaded from the UE to the edge server is p. According to the Shannon formula, the communication rate of task i can be given by:

$$r_{l,e} = B \cdot \log_2 \left(1 + \frac{p \cdot g^2}{N \cdot B}\right), \tag{3}$$

where B represents the communication bandwidth of the wireless channel between the UE and the BS, and N represents the noise power density of the channel.

3.4. Computational model

In this subsection, we focus on the computational model in terms of response time and system energy consumption of the task, which include edge computing and local computing.

3.4.1. Local computing

The mobile UE at the local layer has certain computing and storage capabilities and can handle some lightweight task requests. The computing capability of the UE is represented by the CPU frequency, which is f_l. According to the task offloading strategy, if task i is assigned to be executed on the UE, then the execution time of task i on the UE and the response time of task i can be expressed as:

$$t_{i,l}^{exe} = \frac{d_i \cdot C}{f_l}, \tag{4}$$

$$t_{i,l}^{res} = t_{i,l}^{ct} - sub_i, \tag{5}$$

where C represents the CPU cycles required by the computing node to process 1 bit of data. $t_{i,l}^{ct}$ represents the completion time of task i, which is the sum of the start execution time of task i and the execution time of task i. The task response time is the difference between the task completion time and the task submission time.

If task i is executed on the UE, there is no communication delay and no communication overhead. The energy consumption overhead of task i on the UE is the execution energy consumption, and the execution energy consumption can be expressed as:

$$E_{i,l}^{exe} = \eta \cdot (f_l)^2 \cdot d_i \cdot C, \tag{6}$$

where $\eta \cdot (f_l)^2$ is the energy consumption of the CPU cycle, η represents the energy factor, and the size depends on the CPU chip architecture [4].

3.4.2. Edge computing

Although the UE has certain computing and storage resources, compared to the edge server, its computing power and storage capacity are relatively limited, and it can only handle a small number of lightweight task requests. To better guarantee the user's QoS request, according to the task offloading strategy, most of the task requests may be offloaded to the edge server for processing. The computing power of the edge server is defined as f_e. Assuming that task i is offloaded to the edge server for processing, the execution time of task i on the edge server and communication time of task i can be expressed as:

$$t_{i,e}^{exe} = \frac{d_i \cdot C}{f_e}, \tag{7}$$

$$t_{l,e}^{tra} = \frac{d_i}{r_{l,e}}. \tag{8}$$

After the task is processed by the edge server, the data size in the result is far less than that before the task is processed, and the downlink rate is higher than the uplink rate. The return delay of the result can be ignored. Therefore, this paper is similar to the previous work on mobile edge computing [12], [23], which ignores the task in the edge server after processing, and the results are returned to the local return delay. Similar to Eq. (5), the response time of task i processed on the edge server is defined as:

$$t_{i,e}^{res} = t_{i,e}^{ct} - sub_i, \tag{9}$$

where $t_{i,e}^{ct}$ is the completion time of task i on the edge server. When task i is offloaded to the edge server for processing, there is communication energy consumption and execution energy consumption on the edge server. Communication energy consumption and execution energy consumption are, respectively, expressed as:

$$E_{l,e}^{tra} = p \cdot t_{l,e}^{tra} = p \cdot \frac{d_i}{r_{l,e}}, \tag{10}$$

$$E_{i,e}^{exe} = q \cdot d_i, \tag{11}$$

where q represents the energy consumption of processing 1 bit of data on the edge server. In summary, the total energy consumption of the system is the sum of the energy consumption of task execution on the local UE, the communication energy consumption of task offloading to the edge server, and the energy consumption of task execution on the edge server.

4. Theoretical background

In this section, the theoretical background of DDQN is described. First, we present a theoretical model of the MDP. Then, we present an elaboration of the details of the DRL approach.

4.1. Markov decision process

MDP [29] is a research theory used to solve sequential decision-making problems in uncertain environments. Sequential decision-making problems refer to problems in which the agent with decision-making ability needs to choose from many actions according to the state of the current environment at each time that obeys the exponential distribution, and the state of the environment may change due to environmental uncertainty. Every time the agent makes a choice and performs an action, the agent receives a timely reward from the external environment. The reward affects the environmental state at the next decision time. In other words, in the current state, after the agent makes a decision and selects an action, it immediately obtains a reward value from the external environment and moves to the next state. With continuous decision-making progress, the state continues to transfer until decision-making ends, and the whole decision-making process is completed. In essence, based on certain decision criteria, the agent continuously explores until it finally selects a set of optimal actions to maximize the long-term reward.

In summary, MDP is a mathematical model for the agent to continuously interact with the external environment, which can be represented by a quadruple (S, A, P, R):

- S represents the state space, which is a collection of nonempty finite states in the MEC scene.
- A represents the action space, which is a collection of all actions that can be selected in the MEC scene. A_s represents the set of all optional actions in the current state s, $A_s = \{a_1, a_2, ..., a_n\}$.
- P represents the state transition probability, and $P\left(s' \mid s, a\right)$ represents the probability of transition to the next state s' after performing action a in the current state s.
- R represents the reward value function, and $R(s, a)$ represents the reward value of the external environment obtained after action a is executed in the current state s.

To reasonably solve the task offloading problem in the MEC environment, this paper models the task offloading problem in the MEC environment as an MDP based on Markov decision theory.

4.2. Deep reinforcement learning

Reinforcement learning (RL) [33] is an important branch of machine learning. In the past few years, a series of RL algorithms have been widely used in cloud computing environments to solve task scheduling and resource allocation problems. However, with the complexity of application scenarios, when RL algorithms deal with high-dimensional state space and action space problems, the defects that easily cause dimensional disasters are gradually increased, and the performance of the algorithm gradually decreases. With the further breakthrough of technology, to compensate for the RL shortcomings, research scholars integrated RL and deep learning (DL) [15] and proposed a new learning method, DRL [20], [24]. DRL gives full play to the advantages of RL and DL. DL is a machine learning structure with multiple hidden layers derived from the study of artificial neural networks. Through a series of transformations on the initial features of a large number of low-dimensional data in DL, the high-level data obtained can represent the relevant category attributes of data to a certain extent and find their distribution rules. In essence, DRL is a kind of method that uses a neural network as a value function estimator, specifically to the MEC scene, that is, fitting a high-dimensional action Q-value through a neural network. The neural network used to fit the Q-value of action is generally called the Q-network, and its learning and fitting process can be expressed as:

$$NN\left(Q\left(s,a;\theta\right)\right)\approx Q^{*}\left(s,a\right). \tag{12}$$

The DDQN algorithm is a specific DRL algorithm that uses a deep neural network (DNN) [14] as a function approximator as a Q-network to fit the Q-value of the action in the scene. The algorithm has two Q-networks with the same structure, one named the current Q-network and the other named the target Q-network. The current Q-network is used to estimate the current Q-value and update the network parameters; the target Q-network is used to calculate the target Q-value, and the target Q-network does not need to update the network parameters. After a certain training time step, the current Q-network will copy its network parameters to the target Q-network, and the update formula of the target Q-value can be expressed as:

$$y_t=r+\gamma\cdot Q\left(s_{t+1},arg\max_{a}Q\left(s_{t+1},a;\theta_t\right);\theta_t^{-}\right), \tag{13}$$

where r is the reward value, γ represents the discount factor, which is the tradeoff between the current reward and the future reward of the agent, and its value ranges from 0 to 1. The larger the value, the greater the agent values the current rewards; in contrast, the agent values the future rewards. θ_t is the parameter of the current Q-network at time t, and θ_t^{-} is the parameter of the target Q-network at time t. $arg\max_{a}Q\left(s_{t+1},a;\theta_t\right)$ represents the action a corresponding to the maximum Q-value obtained by the agent after executing all the decisions at the current moment.

In essence, calculating the target Q-value can be divided into two steps. First, an action is selected based on the current Q-network; then, the action is used to calculate the target Q-value in the target Q-network. This process can also be understood as decoupling the target Q-value selection and the target Q-value calculation. The overestimation of the action Q-value can be eliminated through two steps so that the convergence speed of the algorithm can be accelerated. This process is also the key to improving the DDQN algorithm performance compared with that of other related RL and DRL algorithms.

The update of the current Q-network parameters is coordinated by error backpropagation technology and the gradient descent method according to the square difference loss between the target Q-value and the current Q-value. The loss function can be expressed as:

$$Loss\left(\theta\right)=E\left[\left(y_t-Q\left(s_t,a;\theta_t\right)\right)^2\right]. \tag{14}$$

With continuous training, the loss value is constantly updated and adjusted. When the training reaches a certain number, the loss value is close to 0 and tends to be stable, indicating that the training is sufficient and the algorithm has converged. The pseudocode of the DDQN algorithm is shown in Algorithm 1.

Algorithm 1: The DDQN Online Algorithm.

```
   Input: Scene state s
   Output: Action $a$
 1 Initialize replay memory and set its capacity to $C_{rm}$;
 2 Initialize current Q-network with para $\theta$;
 3 Initialize target Q-network with para $\theta^{-}=\theta$;
 4 for $t=1,T$ do
 5     Select a random action $a_t$ with probability $\varepsilon$; otherwise, select
         $a_t=arg\max_a\left(s_t,a;\theta\right)$;
 6     Execute action $a_t$, obtain reward $r_t$, transfer to the next state $s_{t+1}$;
 7     Store experience tuple $\left(s_t,a_t,r_t,s_{t+1}\right)$ in replay memory;
 8     Sample random minibatch of transitions $\left(s_j,a_j,r_j,s_{j+1}\right)$ from replay
         memory;
 9     Calculate the target Q-value using Eq. (13);
10     Perform a gradient descent step using Eq. (14);
11     Update current Q-network para $\theta$;
12     Every $\xi$ steps, clone current Q-network paras to target Q-network;
13 end
14 Save current Q-network;
15 return $a$;
```

5. Algorithm design

In this section, first, we describe the integrated trust mechanism. Secondly, we give a formal definition of the model optimization problem. Finally, we present the DDTMOA detailed algorithm.

5.1. Integrated trust mechanism

In real scenarios, some criminals may attack the computing node with a virus, causing the computing node to be poisoned, paralyzed and unable to operate normally. Due to the dynamic, open and collaborative nature of the MEC network environment, it is difficult to ensure the security, privacy, and reliability of edge data during task offloading. To alleviate these problems, this paper proposes a task offloading integrated trust evaluation mechanism, which includes trusted identity, trusted behavior, and trusted capability. A certain proportion of untrusted cases will occur randomly in the computing nodes, and the specific proportion is less than or equal to 5%.

5.1.1. Trusted identity

Trusted identity refers to the credibility of the user's identity and the executable credibility of the computing node, and its value is generally binary. The number 1 indicates that the identity is trusted, and 0 indicates that the identity is not trusted. The trusted identity value can be formalized as:

$$V_{ide}=\begin{cases}0,\text{ untrusted identity;}\\1,\text{ trusted identity.}\end{cases} \tag{15}$$

In real life, there may be some malicious intrusion into the network to steal data, and there may also be criminals using viruses to attack the computing node so that the computing node paralysis cannot work normally. Therefore, to be close to reality, this paper considers that users and computing nodes have a certain proportion of untrusted cases. Only when the user's identity is trusted and the computing node is trusted can the user's service request be processed normally by the computing node.

5.1.2. *Trusted behavior*

Trusted behavior refers to the behavior rules that computing nodes must comply with. It is a subjective feeling about the degree of trust in the computing node behavior. It manifests as a subjective feeling in the process of completely distrusting the behavior of the computing node to complete trust. Its value is any value between 0 and 1. The higher the value is, the higher the credibility of the behavior; the lower the value is, the lower the credibility of the behavior.

Trusted behavior is related to the historical and current recorded behavior trust value. This paper uses the historical feedback of the computing node to determine its behavior trust value. Historical feedback is also the historical scheduling result. One of the best reflections of historical scheduling is the response time of task requests, so behavioral trust is time-dependent, and users are more willing to trust the recent computing node scheduling. Therefore, the time decay function is introduced, and the time decay function can be defined as:

$$F\left(n\right)=\frac{t_n-t_{n-1}}{\sum_{j=1}^{n}\left(t_j-t_{j-1}\right)}, \tag{16}$$

where t_j represents the start time of the j-th task request response. The larger $F\left(n\right)$ is, the longer the interval between the start time of this task request response and the start time of the last task request response. In contrast, the shorter the time interval.

Using the characteristics of the limit $\lim_{x\to-\infty}e^x=0$, the time delay factor $\Delta t_n=e^{-F(n)}$ is defined to measure the freshness of the service request. The higher the freshness is, the higher the behavior trust value. The behavior trust value can be defined as:

$$T_n=T_{n-1}\times\Delta t_{n-1}. \tag{17}$$

In this design, the behavior of all computing nodes is considered to be completely trusted at the beginning, that is, the behavior trust value of all computing nodes is initialized to 1, and only when the behavior trust value of computing nodes is higher than the minimum trust value T_n^{low} that can be accepted by the task request, can the task request be responded to.

5.1.3. *Trusted capability*

Trusted capability is the functional attribute displayed by the computing node, and it is a set of QoS-related indicators, for example, computing power, storage space, memory size, deadline, bandwidth and computing node throughput. Only when the relevant attributes of the task request are satisfied can the user's task request respond to the computing node; otherwise, the task request cannot be responded to. The trusted indicators considered in this paper include the computing power, storage space, maximum response time deadline that task execution can tolerate, and security level of the computing node.

In summary, only when the trusted identity, trusted behavior, and trusted capability are all satisfied, can the user's task request be successfully responded to.

5.2. *Problem description*

5.2.1. *Optimization objective*

This paper proposes a new task offloading algorithm based on a comprehensive trusted mechanism and combined with the DDQN algorithm. The purpose of optimization is to ensure the offloading performance of the task while reducing the average response time of the task and the total system energy consumption. The average task response time is the quotient of the sum of the response times of all successfully executed tasks and the total number of successfully executed tasks, and the total energy consumption of the system is the total energy consumption of all tasks processed. Assume the total number of submitted tasks is M, and the number of successfully executed tasks is m. Then, the average response time of the task and the total energy consumption of the system can be expressed as:

$$T_{averes}=\sum_{i=1}^{m}\left(t_{i,l}^{res}+t_{i,e}^{res}\right)/m, \tag{18}$$

$$E_{total}=\sum_{i=1}^{m}\left(E_{i,l}^{exe}+E_{l,e}^{tra}+E_{i,e}^{exe}\right), \tag{19}$$

if the task is executed on the UE, then $t_{i,e}^{res}$, $E_{l,e}^{tra}$, and $E_{i,e}^{exe}$ are 0. Similarly, if the task is offloaded to the edge server for execution, then $t_{i,l}^{res}$ and $E_{i,l}^{exe}$ are 0.

When the whole offloading process is completed, assume that the total number of tasks offloaded to the edge server is Y, and the number of tasks successfully executed on the edge server is y. Then, the task execution success rate and the task offloading success rate can be expressed as:

$$ser=\frac{m}{M}, \tag{20}$$

$$osr=\frac{y}{Y}, \tag{21}$$

where the task offloading success rate and the task execution success rate are two important indicators that reflect the task offloading performance. The higher the task offloading success rate and the task execution success rate, the better the task offloading performance of the MEC system.

5.2.2. *Objective function*

Only successfully executed tasks have time and energy costs. The task offloading integrated trust evaluation mechanism proposed in this paper ensures the security, privacy and reliability of the data during the offloading process. In other words, the integrated trust mechanism is the evaluation mechanism of whether the task can be successfully executed. Only when identity, behavior, and ability are trusted can the task be successfully responded to. Therefore, the objective function can be formulated as:

$$\begin{aligned}
&\min\ T_{averes}\ and\ E_{total}\\
s.t.\ &C1\colon V_{ide}^{user}=1\ and\ V_{ide}^{cn}=1,\\
&C2\colon T_n\geqslant T_n^{low},\\
&C3\colon mem_i\leqslant mem_{cn},\\
&C4\colon cpu_i\leqslant cpu_{cn},\\
&C5\colon t_{i,l}^{res}\leqslant deadline_i,\\
&C6\colon t_{i,e}^{res}\leqslant deadline_i,\\
&C7\colon sct_i\leqslant sct_{cn},
\end{aligned} \tag{22}$$

where V_{ide}^{user} and V_{ide}^{cn} represent the identity trust value of users and computing nodes, respectively, and constraint $C1$ is the formal description of trusted identity. Constraint $C2$ is the formal description of trusted behavior. mem_{cn} and cpu_{cn} represent the storage resources and computing resources of the computing nodes, respectively, and sct_{cn} represents the security level of the computing nodes. The security levels of the edge layer computing nodes considered in this paper are low, medium, and high. Considering that the local layer UEs are generally personal terminal equipment, the security level of UEs is set to high. Constraints $C3$ to $C7$ are the formal description of the trusted capability.

5.2.3. Optimization objective weight estimation

To optimize the two objectives at the same time, a problem worth considering is how to allocate the weight of the two optimization objectives in the scene. At present, some researchers, when considering the biobjective problem, generally divide the weights of the two optimization objectives equally and integrate the two objectives through a subjective weighting method. In this way, considering the weight value of the optimization objective, the subjectivity is too strong and has some limitations, which may affect the convergence result of the algorithm, thereby affecting the optimization performance.

In information theory, entropy is a measure of the degree of chaos in the system. The more chaos there is in a system, the more effective information it contains and the higher the entropy value. Conversely, the smaller the degree of chaos in the system, the smaller the amount of information contained, and the smaller the entropy value. According to the characteristics of entropy, the entropy weight method [38] is regarded as a more objective method for estimating the weight value of a single objective in multiple objectives. Therefore, in this paper, the entropy weight method is used to estimate the weight value of the task average response time and total system energy consumption.

Assuming there are h samples and l indicators in the scene, specifically, there are h computing nodes and l optimization objectives in the scene. The steps for calculating the optimal target weight value using the entropy weight method are as follows:

(1) Since the magnitude of the optimization objectives may not be uniform, it is necessary to standardize the objectives. The min-max standardized processing method is expressed as:

$$x_{i,j} = \frac{x_{i,j} - \min\left\{x_{1,j}, \ldots, x_{h,j}\right\}}{\max\left\{x_{1,j}, \ldots, x_{h,j}\right\} - \min\left\{x_{1,j}, \ldots, x_{h,j}\right\}}. \tag{23}$$

(2) Calculate the proportion of each sample value in the objective under different objectives:

$$p_{i,j} = \frac{x_{i,j}}{\sum_{i=1}^{h} x_{i,j}}. \tag{24}$$

(3) Calculate the entropy of different objectives:

$$e_j = -(\ln(n))^{-1} \sum_{i=1}^{h} p_{i,j} \ln\left(p_{i,j}\right). \tag{25}$$

(4) Calculate information entropy redundancy:

$$er_j = 1 - e_j. \tag{26}$$

(5) Calculate the weight value of the optimization objective:

$$w_j = \frac{er_j}{\sum_{j=1}^{l} er_j}. \tag{27}$$

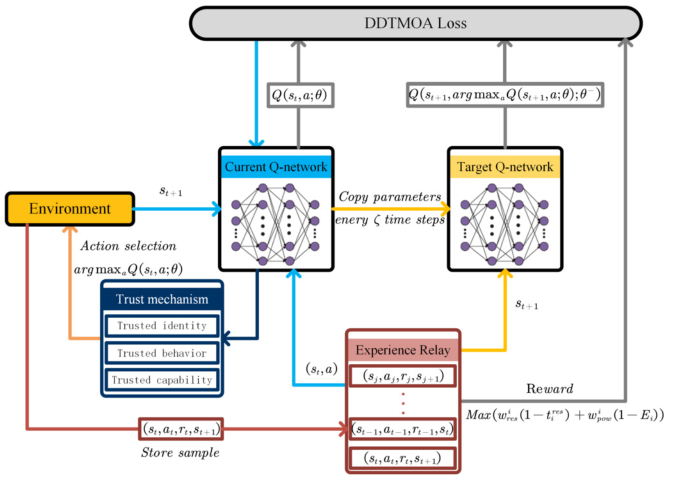

Fig. 2. The DDTMOA framework.

5.3. DDTMOA

This paper proposes a novel task offloading algorithm based on a self-defined integrated trust mechanism combined with the DDQN algorithm, named DDTMOA. The core idea is to use the DL method to fit the state of the scene and then use RL to make a reasonable decision and select the optimal action under the safe and reliable environment of task offloading. To determine whether the task needs to be offloaded, if necessary, determine which edge server the task is offloaded to for processing; if not, determine which UE device the task is allocated to for processing. Finally, the algorithm can effectively reduce the average response time of the task and the total system energy consumption while having a higher task offloading level. The DDTMOA algorithm framework is shown in Fig. 2.

As shown in Fig. 2, the DDTMOA algorithm continuously interacts with the external environment through the Q-network agent. Based on the integrated trust mechanism according to certain learning strategies, the Q-network agent explores the optimal action and executes the optimal decision. Each training generates an experience sample, which is stored in the experience replay pool. The experience samples in the experience replay pool are randomly selected for replay to update the target Q-value. Experience replay technology can increase the learning speed of the algorithm and improve the oscillation and divergence caused by the correlation of experience samples. The state set and action set of the MEC scene and reward function are designed as follows.

State Set. The state of the scene is used as the algorithm input. The state of the scene designed in this paper is simulated by the weighted sum of the response time and energy consumption of the task. The state space of the MEC scene can be expressed as:

$$S = \{s_1, s_2, \ldots, s_i, \ldots, s_h\}, \quad s_i = w_{res}^k \cdot t_i^{res} + w_{pow}^k \cdot E_i, \tag{28}$$

where t_i^{res}, E_i represents the response time and energy consumption of tasks processed on computing node i after standardized processing. w_{res}^k and w_{pow}^k represent the weight values of response time and energy consumption, respectively, in the scene when processing task k.

Action Set. The action set of the scene is a collection of all actions that the agent can choose in each state. Specific to the MEC scenario, the action set is the collection of all computing nodes. The action set of the MEC scene can be expressed as:

$$A = \left\{a_{cn}^1, a_{cn}^2, \ldots, a_{cn}^i, \ldots, a_{cn}^n\right\}, \tag{29}$$

where $a_{cn}^{i} = 1$, which denotes that in this decision-making process, the agent selects action a_{cn}^{i}; that is, the agent assigns the task to computing node i for execution.

Reward Function. Each time the agent executes a decision, it immediately obtains the reward value from the external environment. The reward value is used to evaluate the pros and cons of the action selected by the agent when executing a decision. Obviously, a well-trained Q-network should have the ability to evaluate the pros and cons of the selected actions, affirm reasonable decisions, and deny unreasonable decisions so that the agent can explore maximizing long-term returns. Therefore, the reasonable design of the reward value plays an important role in algorithm performance. The reward value function is designed as:

$$r = \begin{cases} -1, \text{ the constraint does not hold;} \\ w_{res}^{i}\left(1 - t_{i}^{res}\right) + w_{pow}^{i}\left(1 - E_{i}\right), \text{ otherwise} \end{cases} \tag{30}$$

where $r = -1$, which indicates that the agent does not assign an ideal executable computing node to the task during the decision-making process, resulting in the task not responding. In this case, it should be given a negative value and set the worst reward value. When the task can respond, the reward value is set as the weighted sum of the two optimization objectives.

The pseudocode of the DDTMOA algorithm is shown in Algorithm 2.

Algorithm 2: The DDTMOA algorithm.

```
Input: Task set
Output: Task average response time, total energy consumption, task
        offloading success rate, task execution success rate
1  for a task arrives in the task waiting queue do
2      Determine the generation area of the task;
3      Select a computing node for the task using the convergent Q-network
         in Algorithm 1, and based on the integrated trust mechanism;
4      if the selected computing node is UE then
5          Process the task on the local device UE;
6          Calculate the task response time and compute energy consumption;
7      end
8      else
9          Offload the task to an MEC server for processing;
10         Calculate the task response time, compute energy consumption and
             communication energy consumption;
11     end
12 end
13 Calculate the task average response time, total energy consumption, task
     offloading success rate, and task execution success rate;
14 return
```

6. Experiments and results analysis

In this section, extensive experiments are carried out, mainly to evaluate the performance of our proposed DDTMOA algorithm. First, we describe the experimental simulation environment, the MEC scenario parameters, and the Q-network parameters. Second, several classical algorithms that are compared with DDTMOA are presented. Finally, the feasibility and effectiveness of our proposed DDTMOA algorithm are verified through several sets of comparison experiments.

6.1. Simulation environment and experimental parameters

The experimental simulation environment was built on a Python 3.6 platform based on TensorFlow 1.13 under the Windows 10 operating system. Our simulated tasks are generated in real time at the local end, mainly sorting out the real Google data sets. The Google dataset is downloaded from GitHub at https://github.com/google/cluster-data. The original Google dataset contains many task attributes, such as machine ID, platform ID, and logical job name. However, not all of these attributes are required by us, and only some of them are used. In our task model, the task index, task commit time, CPU and memory resources required for execution are extracted directly from within the dataset, while the task data size is obtained based on the cycle time per instruction (CPI) and timestamp calculation. The main simulation parameters in the scenario were designed with similar principles to those in the literature [6], [35], [12], and [23], as shown in Table 1. The relevant hyperparameters in the Q-network are shown in Table 2.

6.2. Comparison algorithms

To verify and evaluate the performance of the task offloading algorithm proposed in this paper, we introduced several classical algorithms to compare with the DDTMOA algorithm. The first is the random offloading algorithm (Random), which refers to the random offloading of tasks to compute nodes for processing and is easy to understand and implement, with low time complexity. It has certain effects in solving tasks such as task scheduling and task offloading. The second is the weighted round-robin offloading algorithm (WRR), which offloads tasks to compute nodes sequentially, and due to the difference in computational power between heterogeneous compute nodes, nodes with higher computational power generally set larger weights. The third and fourth comparison algorithms are task offloading algorithms based on DQL [36] and DQN [26], [22]. DQL and DQN are both specific algorithms in the DRL method, which have been widely used in the MEC environment to solve task offloading and resource allocation problems with relatively satisfactory results, and both can effectively improve MEC system performance. The great difference between these two algorithms is that they have different network structures. DQL has only one deep neural network (DNN), which is used to approximate the action Q-value; DQN approximates the action Q-value through two DNNs with the same structure.

6.3. Performance evaluation

6.3.1. Weight comparison experiments

In this paper, the objective entropy weighting method is used to estimate the weight of different optimization objectives in the MEC scenario. To verify the feasibility, validity and impact on the algorithm performance of the objective entropy weighting method for estimating the weight of optimization objectives, a set of comparison experiments of objective weighting and subjective weighting of the optimization objectives are set up in an MEC scenario with multiple edge regions, multiple heterogeneous edge servers and multiple UE to compare the convergence performance of the DDTMOA algorithm through different weighting methods. The convergence results are shown in Fig. 3 and Fig. 4.

Fig. 3 shows a DDTMOA algorithm convergence graph using the objective entropy weighting method to estimate the optimization objective weights. Fig. 4 shows the DDTMOA algorithm convergence graph obtained by optimizing the objective weights by the subjective mean. The subjective mean weighting approach, which is also a weighting approach that is currently the focus of some research scholars [27], [3], has certain limitations. From Fig. 3 and Fig. 4, it is clear that the DDTMOA algorithm converges faster and the final convergence result is more stable when using the entropy weighting method to estimate the optimization objective weights, which indicates that it is feasible and more effective to use the entropy weighting method to estimate the optimization objective weights.

To further verify whether the entropy weight method can improve the performance of the algorithm. We compare the perfor-

Table 1
Experimental Simulation Parameters.

Notation	Description	Value
C (cycle)	The CPU cycles required to process 1 bit data	500
f_l (GHz)	The computing power of UE	$Unif$ (0.5,1.0)
f_e (GHz)	The computing power of edge servers	$Unif$ (5.0,10.0)
P (W)	The transmit power	0.5
q (J)	The energy consumption of edge servers to process 1 bit of data	$Unif\left(1.0\times10^{-9}, 2.0\times10^{-9}\right)$
B (MHz)	The communication bandwidth	2.0
N (W/Hz)	The noise power density of the channel	10^{-12}
η	The energy factor	10^{-28}
D (km)	The communication distance between UEs and BSs	[0.1,0.2,0.3]

Table 2
Q-network Parameter Settings.

Parameters	Value
The greedy coefficient ε	0.5
The discount factor γ	0.9
The learning rate α	0.01
The minibatch size Δ	32
Target Q-network parameter copy frequency ζ	20
The activation function	ReLU
The gradient optimizer	AdaDelta
The loss function	Mean square error

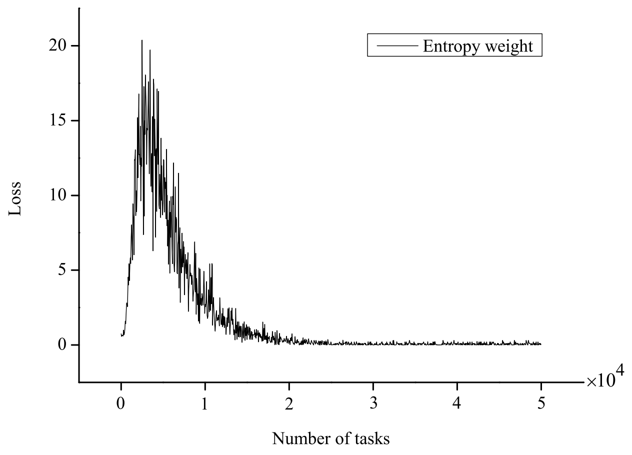

Fig. 3. The objective entropy weight method DDTMOA algorithm convergence graph.

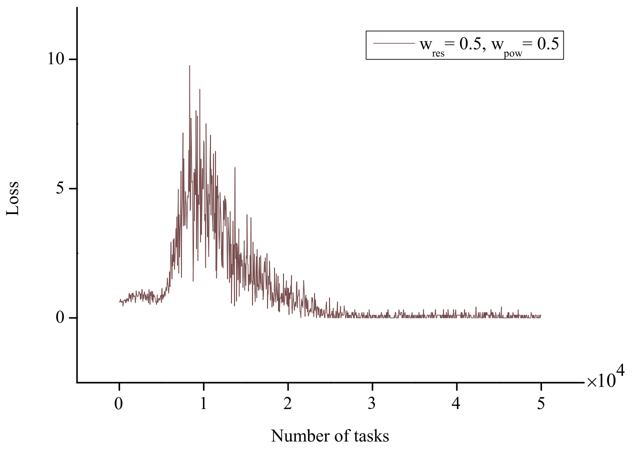

Fig. 4. The subjective weight method DDTMOA algorithm convergence graph.

mance of the DDTMOA algorithm when the optimization objective takes the subjective weight values and the objective entropy weight method to estimate the optimization objective weight values. Three sets of different subjective weight values are set: 1. the weight value of task response time is 0.5, and the weight value of energy consumption is 0.5 $\left(w_{res}=0.5, w_{pow}=0.5\right)$; 2. the weight value of task response time is 0.4, and the weight value of energy consumption is 0.6 $\left(w_{res}=0.4, w_{pow}=0.6\right)$; 3. the weight value of task response time is 0.6, and the weight value of energy consumption is 0.4 $\left(w_{res}=0.6, w_{pow}=0.4\right)$. The experimental results are as follows:

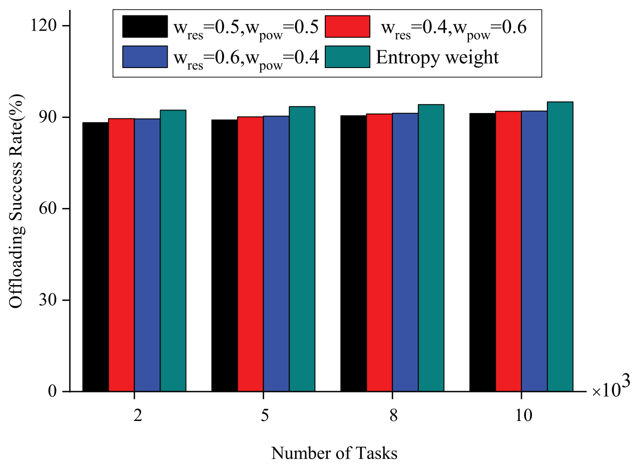

Fig. 5. Task offloading success rate.

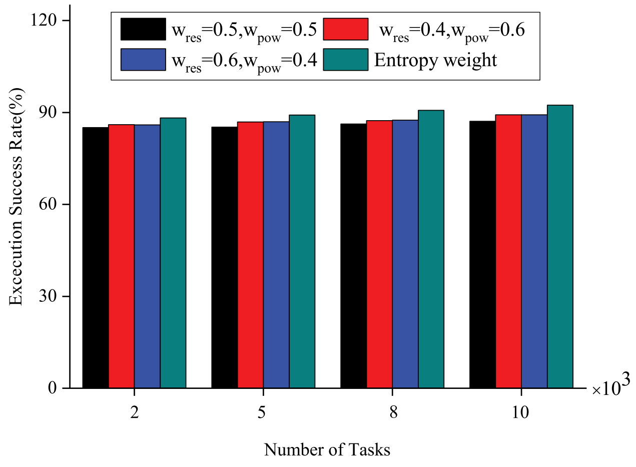

Fig. 6. Task execution success rate.

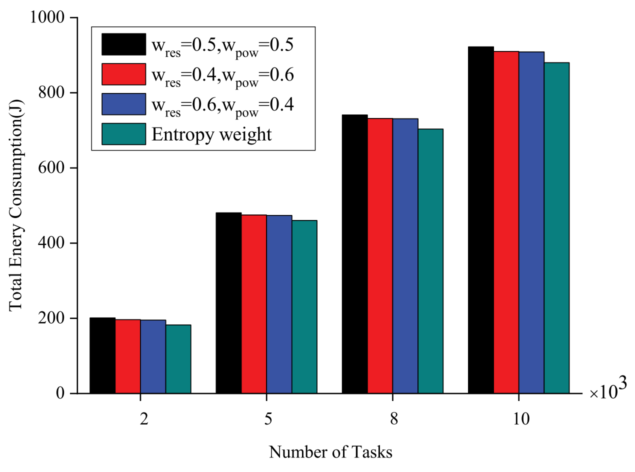

Fig. 7. Total system energy consumption.

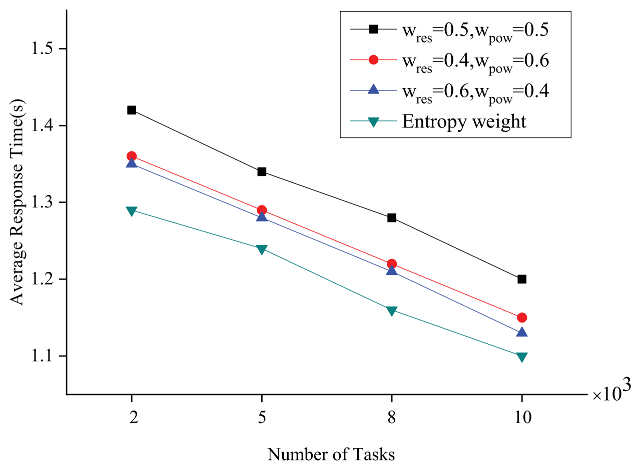

Fig. 8. Average task response time.

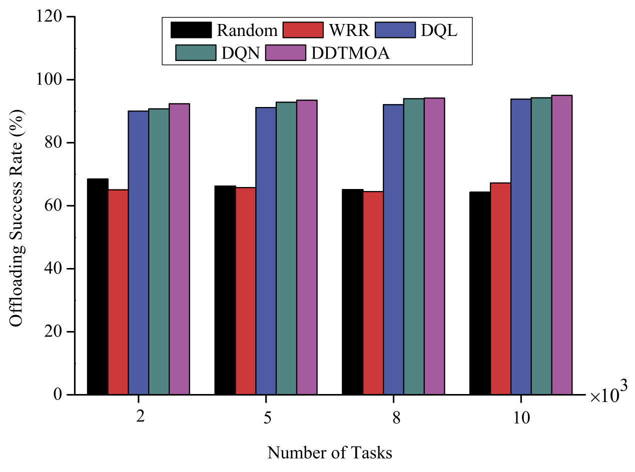

Fig. 9. Task offloading success rate.

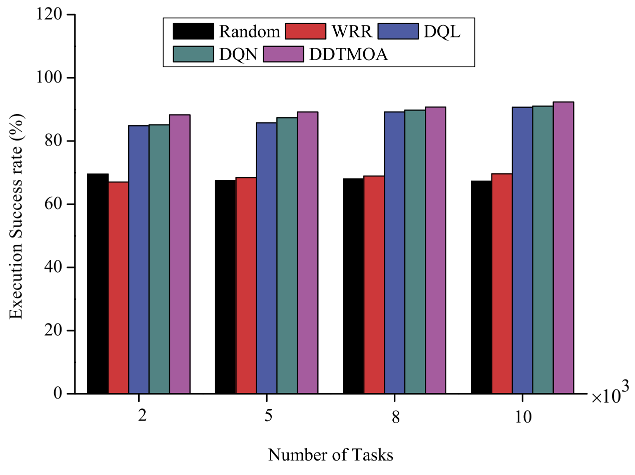

Fig. 10. Task execution success rate.

As shown in Fig. 5, Fig. 6, Fig. 7 and Fig. 8, when the objective entropy weight method is used to estimate the optimization objective weight values, the task offloading success rate and task execution success rate of the DDTMOA algorithm are smaller, and it has lower energy consumption and a smaller task average response time, which further shows that the algorithm has better performance when estimating the optimization target weight values by the objective entropy weight method.

6.3.2. *Basic experiments*

To verify DDTMOA algorithm performance, this section conducts comparative experiments based on different numbers of task sets in an MEC scenario with 9 heterogeneous edge servers and 18 heterogeneous UE devices. That is, in the basic experiment, we designed the number of edge servers, BSs and UEs to be the same as in Fig. 1. The experimental results are as follows:

As shown in Fig. 9, the DDTMOA algorithm always maintains a high task offloading success rate under different numbers of task sets, the task offloading success rate of the DQN and DQL algorithms is slightly lower than that of the DDTMOA algorithm, and the task offloading success rate of the WRR algorithm is lower than that of the DQN and DQL algorithms but higher than that of the random algorithm. This is because the random algorithm has greater randomness, randomly offloading tasks to different edge servers for processing, which can easily lead to violations under numerous trusted constraints, thus affecting the task offloading success rate. The core idea of the WRR algorithm is to assign different weight values to the computing nodes according to their computing power. The nodes with stronger computing power set relatively larger weight values so that more tasks are offloaded to the computing nodes with more sufficient resources. DQN and DQL are two specific DRL algorithms that have strong adaptive learning capabilities and can make decisions more rationally. Therefore, they have a high offloading success rate. However, the DDTMOA algorithm is a further optimization based on both DQN and DQL. Therefore, the DDTMOA algorithm has a high offloading success rate and is relatively stable.

As shown in Fig. 10, the DDTMOA algorithm also always has a high task execution success rate for different sets of tasks. The task execution success rate includes both cases where the task is offloaded to the edge and where the task is executed locally for successful execution. To ensure that the task can respond successfully, the DDTMOA algorithm eventually explores a set of optimal policies through adaptive learning to select the optimal action and reasonably assign the task to a more resource-rich computing node for processing. As a result, the proportion of tasks allocated to resource-poor UE will be relatively small, the default rate will be relatively low, and the task execution success rate will be relatively high.

As shown in Fig. 11, the DDTMOA algorithm maintains low total system energy consumption as the number of tasks continues to increase. This indicates that the DDTMOA algorithm can make decisions more rationally and can reasonably offload tasks based on

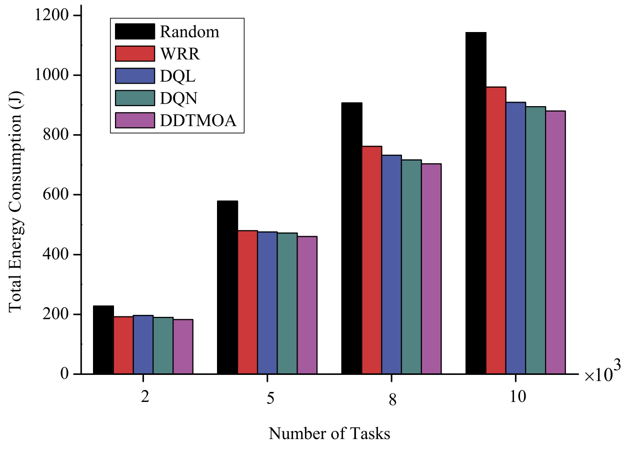

Fig. 11. Total system energy consumption.

the number of resources requested for task execution and the resources of the computing nodes. That is, tasks with relatively large quantities of data are offloaded to compute nodes with more sufficient computational resources, while tasks with relatively small quantities of data are offloaded to compute nodes with fewer computational resources. This ensures that the tasks can be successfully responded to and that the computing resources can be fully utilized, thus enabling the battery life of the UE on the local end. Combined with Fig. 10, it can be seen that the DDTMOA algorithm can maintain a high task execution success rate and has low energy consumption, which further illustrates the advantages of the DDTMOA algorithm over other algorithms.

It is not easy to ensure that no excessive system energy is consumed and that the task has a low response time. However, algorithm designs that sacrifice one goal to achieve another goal are not reasonable. Based on different numbers of task sets, the comparison results of the average response time of each algorithm are as follows:

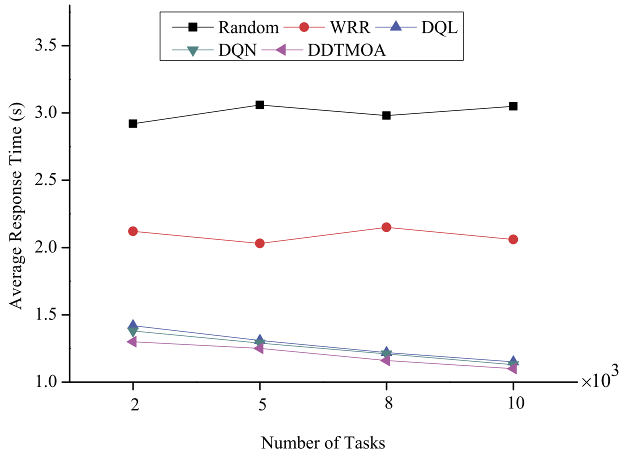

Fig. 12. Average task response time.

As shown in Fig. 12, the DDTMOA algorithm has a low average task response time as the number of tasks increases, which indicates that when using the DDTMOA algorithm, more tasks are offloaded to computational nodes with more computing resources and computational power for execution, thus reducing the waiting time and execution time of tasks resulting in a more significant decrease in the overall task response time. Combined with Fig. 11, it can be seen that the DDTMOA algorithm better balances the two optimization objectives of total system energy consumption and average task response time compared to other algorithms.

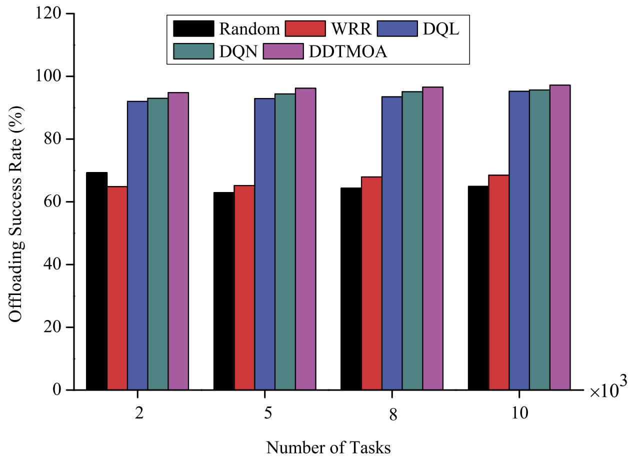

Fig. 13. Task offloading success rate.

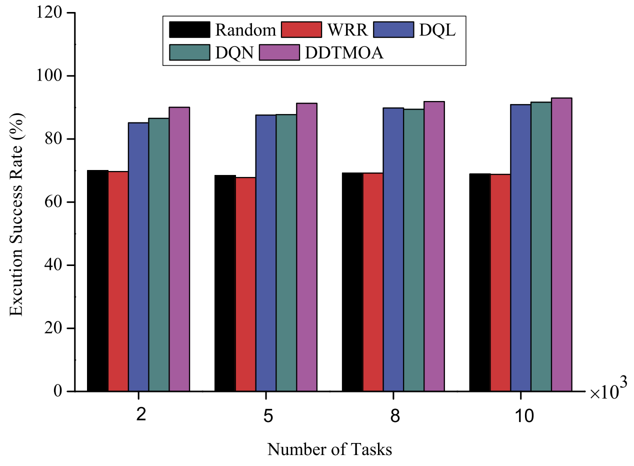

Fig. 14. Task execution success rate.

6.3.3. *Extension experiments*

To further validate the scalability and stability of the DDTMOA algorithm, the MEC scenario simulated in this section was extended by increasing the number of UE devices at the local end from 18 to 50. Comparative experiments were conducted under different numbers of task sets based on the multiple UE scenario.

As shown in Fig. 13 and Fig. 14, the DDTMOA algorithm maintains a low task offloading success rate and task execution success rate for different numbers of task sets as the number of compute nodes increases. This further shows that based on the more complex MEC scenario, the DDTMOA algorithm can make more rational decisions than other algorithms and can more effectively determine whether a task needs to be offloaded and to which edge server it should be executed, thus ensuring that the task can be offloaded and executed successfully.

The battery life of UE devices at the local end is limited, and their computing power is relatively weak. Too many tasks assigned to the UE for execution may both consume resources and fail to meet the user's QoS requirements. Task offloading to the edge server not only has execution energy consumption but also generates additional communication energy consumption. Therefore, reasonable task offloading to reduce system energy overhead is particularly important. As shown in Fig. 15, the DDTMOA algorithm

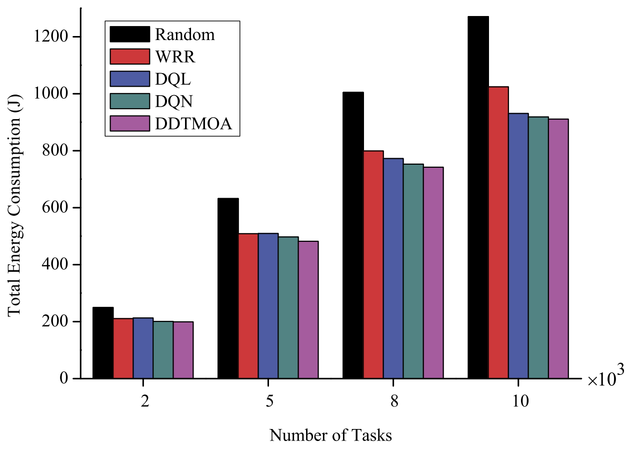

Fig. 15. Total system energy consumption.

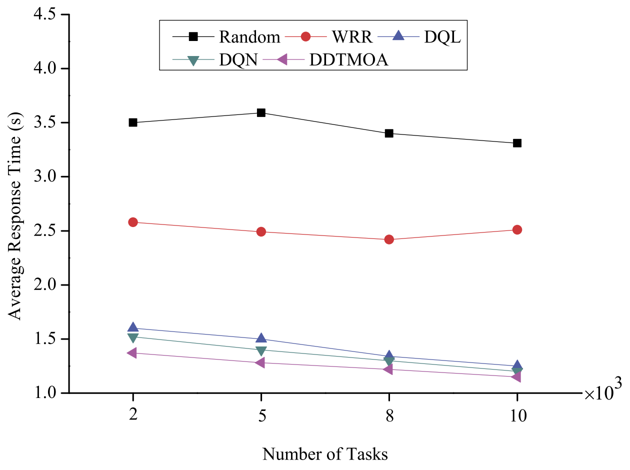

Fig. 16. Average task response time.

maintains a low total system energy consumption for different numbers of task sets, which indicates that the DDTMOA algorithm can fully and reasonably utilize the various resources of the computing nodes compared to other algorithms to ensure the energy consumption level of the system.

As shown in Fig. 16, the DDTMOA algorithm also always has a low average task response time and is relatively stable under different numbers of task sets. Combined with Fig. 15, it is easy to see that the DDTMOA algorithm is more effective than other algorithms in balancing the energy and time overheads of the system and can reduce both the total energy consumption and the average response time of the tasks. This shows that DDTMOA is more intelligent than other algorithms and can make more reasonable decisions based on the resources requested by the tasks and the resources of the computing nodes.

To more intuitively reflect the robustness and stability of the algorithm, this section also compares the standard deviation before and after the task offloading success rate, the average task energy consumption, and the average task response time of the basic experiment and the extended experiment. The comparison results are shown in Figs. 17, 18, and 19.

As seen in Figs. 17, 18, and 19, the standard deviation of the task offloading success rate, the standard deviation of the task average energy consumption and the standard deviation of the task average response time for both the basic and extended experiments of the DDTMOA algorithm are small compared to the other algorithms. Moreover, based on the overall view, the difference

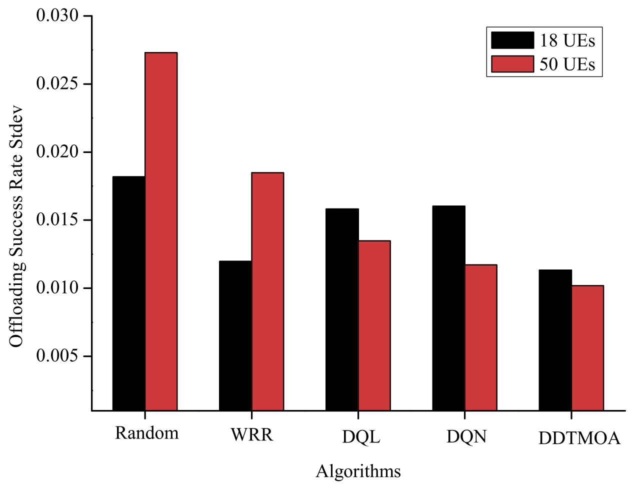

Fig. 17. Task offloading success rate standard deviation comparison.

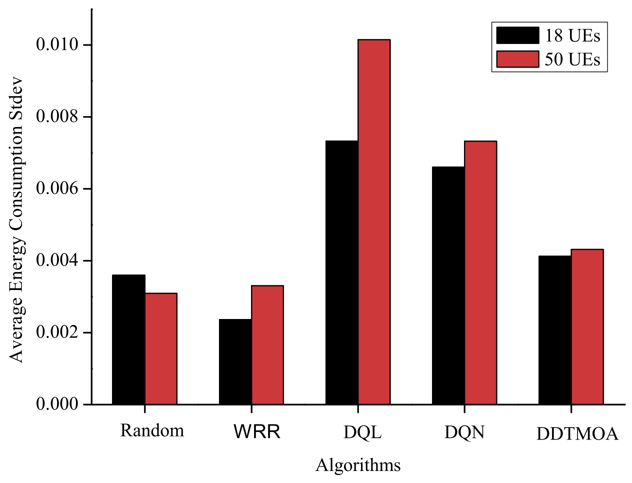

Fig. 18. Average task energy consumption standard deviation comparison.

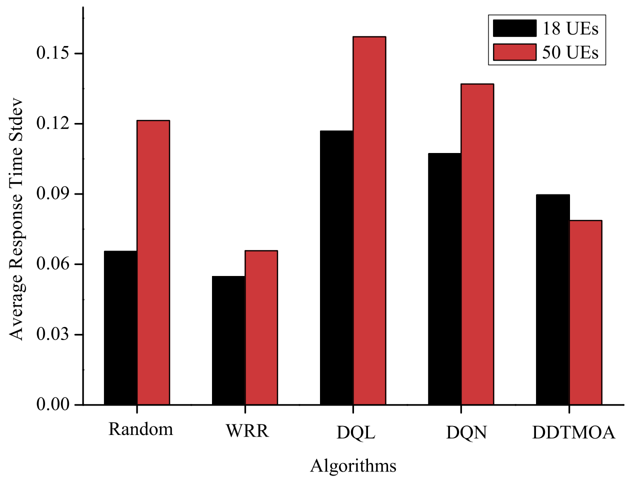

Fig. 19. Average task response time standard deviation comparison.

between the standard deviation of each metric of the DDTMOA algorithm in the basic and extended experiments is also small. This indicates that the fluctuations of the DDTMOA algorithm are minimal as the number of computational nodes increases, verifying that the DDTMOA algorithm has good stability and robustness.

Based on all the above comparison experiments, it can be seen that the DDTMOA algorithm has better offloading performance than other algorithms, can effectively reduce the average task response time and the total system energy consumption, and has good scalability, robustness and stability.

7. Conclusions and future work

This paper focuses on the problem of task offloading in the MEC environment. First, a two-tier, multiheterogeneous edge server, multiple UE device MEC scenario is constructed. Second, combined with the DDQN algorithm, we propose the DDTMOA. We consider the security of offloading tasks in the MEC environment and construct a comprehensive trust evaluation mechanism for task offloading by discriminating the trusted identity, trusted behavior and trusted capability of users and computing nodes. We use a biobjective to train the algorithm for optimization, so we use an entropy weighting method to dynamically determine the weights of latency and energy consumption. Finally, we conducted basic and extended experiments to verify the effectiveness and robustness of the algorithm using a real Google dataset.

In future work, we will consider more complex MEC scenarios, introduce cloud centers into MEC scenarios, realize cloud-edge collaboration, and better improve the MEC system performance. Additionally, more comprehensive data communication methods will be considered.

CRediT authorship contribution statement

Zhao Tong: Idea, Survey, Optimization, Reviewing.
Feng Ye: Survey, Implement, Experiments, Writing - original draft.
Jing Mei: Optimization, Reviewing.
Bilan Liu: Suggestions, Reviewing.
Keqin Li: Suggestions, Reviewing.

Declaration of competing interest

We wish to draw the attention of the Editor to the following facts which may be considered as potential conflicts of interest and to significant financial contributions to this work.

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We further confirm that any aspect of the work covered in this manuscript that has involved either experimental animals or human patients has been conducted with the ethical approval of all relevant bodies and that such approvals are acknowledged within the manuscript.

Acknowledgments

This work was supported by the Program of the National Natural Science Foundation of China (Grant Nos. 62072174, 61502165, 61602170), the National Natural Science Foundation of Hunan Province, China (Grant No. 2020JJ5370), and the National Outstanding Youth Science Program of the National Natural Science Foundation of China (Grant No. 61625202).

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: a survey, IEEE Int. Things J. 5 (1) (2017) 450–465.
[2] Y. Ai, M. Peng, K. Zhang, Edge computing technologies for internet of things: a primer, Digital Communications and Networks 4 (2) (2018) 77–86.
[3] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, J. Hu, iraf: a deep reinforcement learning approach for collaborative mobile edge computing iot networks, IEEE Int. Things J. 6 (4) (2019) 7011–7024.
[4] K. Cheng, Y. Teng, W. Sun, A. Liu, X. Wang, Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–6.
[5] L. Cui, S. Yang, Z. Chen, Y. Pan, Z. Ming, M. Xu, A decentralized and trusted edge computing platform for internet of things, IEEE Int. Things J. 7 (5) (2019) 3910–3922.
[6] 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.
[7] I.A. Elgendy, W. Zhang, Y.-C. Tian, K. Li, Resource allocation and computation offloading with data security for mobile edge computing, Future Gener. Comput. Syst. 100 (2019) 531–541.
[8] D. Evans, The internet of everything: how more relevant and valuable connections will change the world, Cisco IBSG 2012 (2012) 1–9.
[9] B.S. Gu, L. Gao, X. Wang, Y. Qu, S. Yu, Privacy on the edge: customizable privacy-preserving context sharing in hierarchical edge computing, IEEE Trans. Netw. Sci. Eng. 7 (4) (2019) 2298–2309.
[10] H. He, G. Xu, S. Pang, Z. Zhao, Amts: adaptive multi-objective task scheduling strategy in cloud computing, China Commun. 13 (4) (2016) 162–171.
[11] C.G.C. Index, C. Index, Forecast and Methodology, 2016–2021; White Paper, Cisco Systems, Inc., San Jose, CA, USA, 2017.
[12] C. Kai, H. Zhou, Y. Yi, W. Huang, Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability, IEEE Trans. Cogn. Commun. Netw. (2020), https://doi.org/10.1109/TCCN.2020.3018159.
[13] N. Kiran, C. Pan, S. Wang, C. Yin, Joint resource allocation and computation offloading in mobile edge computing for sdn based wireless networks, J. Commun. Netw. 22 (1) (2019) 1–11.
[14] G. Li, S.K.S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, S.W. Keckler, Understanding error propagation in deep learning neural network (dnn) accelerators and applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2017, pp. 1–12.
[15] H. Li, K. Ota, M. Dong, Learning iot in edge: deep learning for the internet of things with edge computing, IEEE Netw. 32 (1) (2018) 96–101.
[16] T. Liu, Y. Zhang, Y. Zhu, W. Tong, Y. Yang, Online computation offloading and resource scheduling in mobile edge computing, IEEE Int. Things J. 8 (8) (2021) 6649–6664.
[17] Y. Liu, H. Yu, S. Xie, Y. Zhang, Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks, IEEE Trans. Veh. Technol. 68 (11) (2019) 11158–11168.
[18] H. Lu, C. Gu, F. Luo, W. Ding, X. Liu, Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning, Future Gener. Comput. Syst. 102 (2020) 847–861.
[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, Comput. Sci. (2013).
[20] 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.
[21] M. Mukherjee, R. Matam, C.X. Mavromoustakis, H. Jiang, M. Guo, Intelligent edge computing: security and privacy challenges, IEEE Commun. Mag. 58 (9) (2020) 26–31.
[22] Z. Peng, J. Lin, D. Cui, Q. Li, J. He, A multi-objective trade-off framework for cloud resource scheduling based on the deep q-network algorithm, Clust. Comput. 10 (2020).
[23] J. Ren, G. Yu, Y. He, G.Y. Li, Collaborative cloud and edge computing for latency minimization, IEEE Trans. Veh. Technol. 68 (5) (2019) 5031–5044.
[24] Z. Tong, F. Ye, M. Yan, H. Liu, S. Basodi, A survey on algorithms for intelligent computing and smart city applications, Big Data Min. Anal. 4 (3) (2021) 155–172.

[25] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, 2016, pp. 2094–2100.
[26] M. Volodymyr, K. Koray, S. David, A.A. Rusu, V. Joel, M.G. Bellemare, G. Alex, R. Martin, A.K. Fidjeland, O. a Georg, Human-level control through deep reinforcement learning, Nature 518 (7540) (2019) 529–533.
[27] J. Wang, L. Zhao, J. Liu, N. Kato, Smart resource allocation for mobile edge computing: a deep reinforcement learning approach, IEEE Trans. Emerg. Top. Comput. (2019), https://doi.org/10.1109/TETC.2019.2902661.
[28] K. Wang, K. Yang, C.S. Magurawalage, Joint energy minimization and resource allocation in c-ran with mobile cloud, IEEE Trans. Cloud Comput. 6 (3) (2016) 760–770.
[29] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, K.K. Leung, Dynamic service migration in mobile edge computing based on Markov decision process, IEEE/ACM Trans. Netw. 27 (3) (2019) 1272–1288.
[30] T. Wang, H. Luo, X. Zheng, M. Xie, Crowdsourcing mechanism for trust evaluation in cpcs based on intelligent mobile edge computing, ACM Trans. Intell. Syst. Technol. 10 (6) (2019) 1–19.
[31] X. Wang, K. Wang, S. Wu, S. Di, H. Jin, K. Yang, S. Ou, Dynamic resource scheduling in mobile edge cloud with cloud radio access network, IEEE Trans. Parallel Distrib. Syst. 29 (11) (2018) 2429–2445.
[32] L. Xi, L. Yu, Y. Xu, S. Wang, X. Chen, A novel multi-agent ddqn-ad method-based distributed strategy for automatic generation control of integrated energy systems, IEEE Trans. Sustain. Energy 11 (4) (2019) 2417–2426.
[33] J. Yang, X. You, G. Wu, M.M. Hassan, A. Almogren, J. Guna, Application of reinforcement learning in uav cluster task scheduling, Future Gener. Comput. Syst. 95 (2019) 140–148.
[34] Z. Yang, Y. Liu, Y. Chen, N. Al-Dhahir, Cache-aided noma mobile edge computing: a reinforcement learning approach, IEEE Trans. Wirel. Commun. 19 (10) (2020) 6899–6915.
[35] J. Zhang, X. Hu, Z. Ning, E.C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, IEEE Int. Things J. 5 (4) (2017) 2633–2645.
[36] Q. Zhang, M. Lin, L.T. Yang, Z. Chen, P. Li, Energy-efficient scheduling for real-time systems based on deep q-learning model, IEEE Trans. Sustain. Comput. 4 (1) (2017) 132–141.
[37] R. Zhao, X. Wang, J. Xia, L. Fan, Deep reinforcement learning based mobile edge computing for intelligent internet of things, Phys. Commun. 43 (2020) 101184.
[38] Y. Zhu, D. Tian, F. Yan, Effectiveness of entropy weight method in decision-making, Math. Probl. Eng. 2020 (2020) 1–5.

Zhao Tong received the PhD degree in computer science from Hunan University, China in 2014. He was a visiting scholar at the Georgia State University from 2017 to 2018. He is currently an associate professor at the College of Information Science and Engineering of Hunan Normal University. His research interests include modeling and scheduling for parallel and distributed computing systems. He has published more than 15 research papers in international conferences and journals, such as IEEE Transactions on Parallel and Distributed Systems, Information Sciences, Neural Computing and Applications, and Journal of Parallel and Distributed System. He is a member of CCF.

Feng Ye received the BE degree from Hunan Institute of Science and Technology, Yueyang, China in 2018. He is currently a master student at the College of Information Science and Engineering, Hunan Normal University, Changsha, China. His research interests include cloud computing, mobile edge computing, objective optimization, task scheduling, machine learning, and artificial intelligence.

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

Bilan Liu received the BE degree from Hunan Institute of Science and Technology, Yueyang, China in 2020. She is currently pursuing a MS degree in the College of Information Science and Engineering of Hunan Normal University, Changsha, China. Her research interests include Cloud-edge 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.