Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 6107893, 9 pages
https://doi.org/10.1155/2022/6107893

Research Article

Multiuser Computing Offload Algorithm Based on Mobile Edge Computing in the Internet of Things Environment

Xiao Chu and Ze Leng

Changchun University of Finances and Economics, Changchun, Jilin 130122, China

Correspondence should be addressed to Xiao Chu; chuxiao1983@ccufe.edu.cn

Received 5 January 2022; Revised 29 January 2022; Accepted 31 January 2022; Published 3 March 2022

Academic Editor: Shalli Rani

As traditional cloud computing is not efficient enough to support large-scale computational task execution in IoT environments, a task offloading and resource allocation algorithm for mobile edge computing (MEC) is proposed in this paper. First, a multiuser computation offloading model is constructed, including a communication model and computation offloading model, which is transformed into the minimization of users' time delay and energy consumption (i.e., total system overhead) in the MEC system. Then, the task offloading model is formulated into a Markov decision process, and an offloading strategy based on a deep Q network (DQN) is designed to dynamically make fine tunings on the offloading proportion of each user so as to realize a low-cost MEC system. The proposed algorithm is analyzed based on the constructed simulation platform. The simulation results show that when the number of user terminals is 40, the average delay of the proposed algorithm does not exceed 0.9 s, and the average energy consumption tends to 65 J, which is better than the comparison method. Therefore, the proposed algorithm has certain application prospects.

1. Introduction

Nowadays, more and more mobile devices are emerging in people's lives, leading to the explosion in the population of smart network edge devices [1]. Subsequently, the data and computation tasks are also growing exponentially. In the context of massive data and large-scale computation tasks, mobile devices are required to process large amounts of application data quickly, which lays a high demand on their computing capacity. Due to the mismatch between data volume and transmission channel, traditional cloud computing brings huge pressure and ultrahigh delays to the crowded network and cannot efficiently support the execution of large-scale computing tasks [2]. Mobile edge computing (MEC) provides cloud computing capacity for mobile devices at the edge of networks via wireless access, which solves the problem of limited computation and energy resources for mobile devices. MEC has become a new paradigm for providing powerful computing and storage capabilities for mobile devices [3, 4]. In order to further ameliorate the quality of services for users and the increase of resource utilization efficiency in MEC, complex computation task offloading strategies and the allocation of communication resources need to be addressed [5].

In early work on computation offloading, most researches considered single-user scenarios, such as low complexity dynamic computation offloading algorithms based on Markov decision processes and Lyapunov optimization, trying to achieve the load-balancing optimization through offloading strategies [6, 7]. Reference [8] proposed a low complexity heuristic algorithm to achieve load balancing by using fractional programming with the optimization goal of minimizing the energy consumption of task offloading. However, the multiscenario and multidimensional optimization for computational resource allocation is yet to be improved. Reference [9] realizes task unloading and efficient channel resource allocation based on the differential evolution algorithm. This scheme can significantly reduce energy consumption while ensuring convergence. However, the performance is poor for multiobjective optimization of

complex tasks. On the new cloud edge computing network designed by Reference [10], a joint optimization strategy based on a binary custom fireworks algorithm is proposed, which can ensure the rationality of system computing resources and response time. But the high occupancy of computational resources and the resource utilization need to be improved. Reference [11] proposed a joint distributed algorithm considering transmission power and unloading strategy and established a queue model with a separate capacity between different windows to optimize queue delay. However, with the massive number of network devices accessing the network in the 5G era, single-user scenarios are no longer able to meet people's daily needs.

Recently, deep learning techniques have been widely studied with the development of artificial intelligence. Since deep learning can solve some limitations in reinforcement learning, it is integrated into reinforcement learning to open a new era of deep reinforcement learning [12]. Deep reinforcement learning incorporates deep neural networks to optimize the process of reinforcement learning, thus improving the learning speed and performance of reinforcement learning algorithms. Therefore, deep reinforcement learning is widely used in the practice of reinforcement learning [13]. Reference [14] proposed a distributed optimization method based on an alternating direction multiplier, which decomposes the optimization problem into N subproblems and maximizes the weighted sum calculation rate through the optimal allocation of system resources and task calculation time. However, this method is weakly adaptable to new environments. An offloading strategy based on metareinforcement learning was proposed in Reference [15]. Mobile applications are modeled as directed acyclic graphs and offloading strategies via neural networks, and the collaboration of first-order approximation and tailoring of agent goals is applied for effective training. Although the adaptability is enhanced, the processing time for the strategy still needs to be improved. Reference [16] constructs a task unloading model based on multiagent deep reinforcement learning and uses the MEC model to better realize computing task unloading and resource allocation. Wang et al. proposed a reinforcement learning-based computing offloading strategy [17]. Although the aforementioned deep learning algorithms can achieve better performance in MEC, the training process and initial conditions are very complex, which means they need to be further optimized in practical applications.

Based on the above analysis, to alleviate the network load and reduce the risk of network congestion in traditional cloud computing of IoT, MEC is introduced to formulate the multiuser computing offloading problem. A task offloading and resource allocation algorithm for MEC in an IoT environment is proposed. Since user's tasks and the computation tasks in the edge server may be time-varying, a deep Q network (DQN) based computation offloading strategy is proposed to achieve the minimum operation overhead of the system by dynamically fine-tuning the ratio of time delay and energy consumption, which improves the robustness of the proposed algorithm.

2. System Model and Optimization Objectives

2.1. System Model. In the MEC system, in order to better serve users and improve system task processing capacity, computing tasks can be offloaded to the MEC server for execution via the wireless channel according to practical situations [18]. As shown in Figure 1, the number of mobile users is $n = \{1, 2, \cdots, N\}$. The MEC server is deployed in the system, which is connected to the base station of this cell. Namely, in the process of task offloading, the computation task of each user cannot be split but can only be offloaded. Meanwhile, the offloading strategy cannot be changed.

Supposing that the number of wireless transmission channels between users and the base station is $m = \{1, 2, \cdots, M\}$, users can choose one of the multiple wireless channels to offload tasks. The offloading strategy of the user n can be denoted as $a_n = \{0, 1, \cdots, M\}$. When $a_n = 0$, it indicates that the user selects local computing, and when $a_n > 0$, it indicates that the user selects to uninstall the MEC server for execution.

Assume that the computation-intensive task is $T_n = \{d_n, c_n\}$. Where d_n and c_n denote the input data size of the task in K bits and the CPU cycles required to process the input data, respectively.

2.2. Communication Model. The communication model of the system includes the selection and assignment of the channel. When a user is connected n to the server Ω, let $S^a_{n,\Omega} = 1$, then the user n can offload tasks T_n to the server through the high-speed network. At this point, if a task needs to be offloaded, the server is required to allocate a certain amount of network bandwidth to the user, which is denoted as $B_{n,i}$. Since the offloading time for a task is very short, we assume that the bandwidth obtained by the task is not grabbed during the offloading. When the task is offloaded, the occupied bandwidth will be released and the total bandwidth provided by the server Ω is B_Ω. The server can choose different bandwidth allocation methods β^C_Ω, such as fixed percentage allocation, fixed amount allocation, or allocation based on user payment criteria.

As the channel selection and allocation strategy are not the focus of this work, the previously proposed communication model is adopted and the bandwidth allocation method is given when several users share a channel that is nonpreemptible. Namely, the bandwidth allocated to a user cannot be released until the user's data have been transmitted [19, 20]. In addition, unlike the strategy that requires waiting for the completion of a communication cycle before releasing the bandwidth, the occupied bandwidth is released immediately after the transmission is completed in this paper. And it is shown in this paper that immediate bandwidth release helps to improve the bandwidth utilization of the system.

Based on the above analysis, it is known that the remaining bandwidth of the current server is B^a_Ω, and the offloading decision in each round offers n number of users to offload their tasks, then the bandwidth obtained by each user in this offloading process can be written as

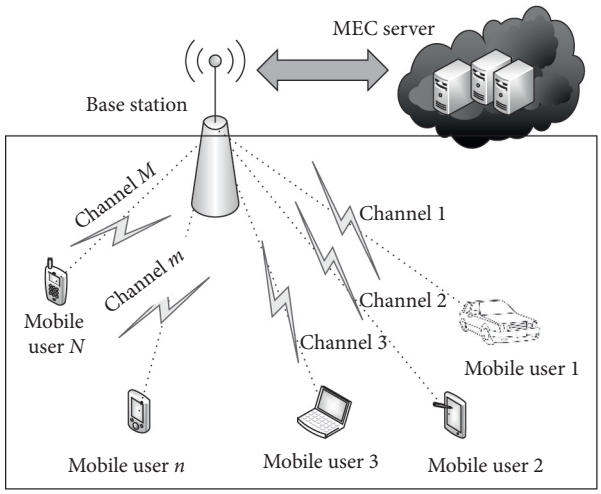

Figure 1: Multiuser MEC system model in single-cell.

$$B_{n,i} = \delta_0 \log_2\left(1 + \frac{p_n \times g_{n,\Omega}}{\delta_0 + \sum_{n \in N}\left(p_{nn} \times g_{nn,\Omega}\right)}\right), \tag{1}$$

where δ_0 is the background noise, p represents the transmission power consumption, and g represents channel gain between the user equipment and the base station.

The model has the following features. If many users choose to offload tasks at the same moment, they will bring large interference among each other and reduce the transmission rate, which results in a challenge for the offloading algorithm when choosing the combination of offloading user devices. For a task with an uplink data volume of $\widehat{d}_{n,i}$, the relationship between the data volume and the transmission time can be calculated as

$$t_{n,i}^{C} = \frac{\widehat{d}_{n,i}}{B_{n,i}}. \tag{2}$$

Thus, the communication model C_Ω of the server can be formulated as

$$C_\Omega = \langle B_\Omega, \beta_\Omega^C \rangle. \tag{3}$$

The communication model $C_{n,i}$ of the user device can be formulated as

$$C_{n,i} = \langle t_{n,i}^C, \widehat{d}_{n,i} \rangle. \tag{4}$$

2.3. Computation Offloading Model

2.3.1. Local Execution. When $a_n = 0$, the user n chooses to execute the computation-intensive task T_n locally. Let f_n be the computing capacity of the user n, then the time delay t_1 incurred by the task T_n when it is executed locally can be calculated as

$$t_1 = \frac{c_n}{f_n}. \tag{5}$$

The energy consumption e_1 for local execution can be calculated as

$$e_1 = \chi_1 c_n \left(f_n\right)^2, \tag{6}$$

where χ_1 is a constant.

According to equations (5) and (6), the total overhead of local execution Θ_1 can be expressed as

$$\Theta_1 = \omega_t t_1 + \omega_e e_1, \tag{7}$$

where ω_t is the weight of time delay, ω_e is the weight of energy consumption, $0 \le \omega_t \le 1$, and $0 \le \omega_e \le 1$, $\omega_t + \omega_e = 1$.

2.3.2. MEC Offloading Computation. When $a_n > 0$, the user selects to uninstall to the MEC server for execution, during the offloading process, the TD and EC are generated during the following three steps: (1) the computation task is transmitting data through the wireless channel; (2) the computation task is executed at the MEC server; and (3) the computation result is returned to the user.

When data are transmitted to the MEC, the user selects the wireless channel, and the resulting time delay can be written as

$$t_2 = \frac{d_n}{\widehat{v}_n(a)}, \tag{8}$$

where $\widehat{v}_n(a)$ is the uplink data transmission rate.

The energy consumption incurred by the data transmission to the MEC can be expressed as

$$e_2 = p_n t_2 = \frac{p_n d_n}{\widehat{v}_n(a)}. \tag{9}$$

When the task is uploaded to the MEC server, it is computed using the computational resources of the server, at this point the resulting time delay can be calculated as

$$t_3 = \frac{c_n}{F_n}, \tag{10}$$

where F_n denotes the MEC server computing capacity.

The energy consumed when executing tasks at the MEC can be formulated as

$$e_3 = \chi_0 c_n \left(F_n\right)^2. \tag{11}$$

Generally, the size of the result calculated by the MEC server is very small compared with the input data, so the TD and EC when returning the calculation result to the user can be ignored [21]. Thus, the total overhead of offloading the computation task to the MEC for execution can be expressed as

$$\Theta_2 = \omega_t\left(t_2 + t_3\right) + \omega_e\left(e_2 + e_3\right). \tag{12}$$

Based on equations (7) and (11), the overhead of each user can be expressed as

$$\Theta_n(a) = \begin{cases} \Theta_1, & a_n = 0, \\ \Theta_2, & a_n > 0. \end{cases} \tag{13}$$

2.4. Optimization Objectives. The optimization objective of a multiuser MEC system is to minimize the TD and EC. Hence, it can be modeled as

$$\begin{aligned} &\min \sum_{n \in N} \Theta_n(a) \\ &\quad a_n \in \{0, 1, \cdots, M\}, \forall n \in N \\ &\text{s.t.} \\ &\quad p_n \in [p_{\min}, p_{\max}], \end{aligned} \tag{14}$$

where $p_{\min}$ is the minimum values of the transmission power and $p_{\max}$ is the maximum values of the transmission power.

The above optimization problem involves the combinatorial optimization problem in multidimensional discrete space. We can consider using reinforcement learning technology and making use of the intelligent characteristics of mobile users so that mobile users can get mutually satisfactory unloading strategies.

3. Solutions

3.1. Reinforcement Learning. Reinforcement learning (RL) is an autonomous learning framework that implements experience-driven learning through interactions and is used to maximize the reward when intelligent agents are finding the optimal behavior at a given state. Reinforcement learning, as a part of machine learning, differs from supervised learning, where training is based on the right answer itself [22, 23]. In a standard RL model, the autonomous-learning agents interact with the environment. The process of reinforcement learning is shown in Figure 2. At each timestamp t, a state $s(t)$ is first observed from the environment, then an action $\varphi(t)$ is executed based on the current state, after which a reward/punishment r_t is fed back by the current environment. Thereafter, the environment will move to another state $s(t+1)$, where the probability of the environment moving to a state $s(t+1)$ after performing an action $\varphi(t)$ from the state $s(t)$ can be represented by the state probability transfer function $\sigma(s(t+1)|s(t), \varphi(t))$.

The process described above is going to continue, which maximizes the desired reward in the long run. Mathematically, reinforcement learning can be described as a Markovian decision process in which the response of the environment to the state $s(t+1)$ depends on $s(t)$ and $\varphi(t)$. Furthermore, the key point of reinforcement learning is to learn without the knowledge of the underlying environment model. And the reinforcement learning that cannot compute rewards before actions are selected and cannot know the state probability transfer function is referred as model-free reinforcement learning [24]. Meanwhile, reinforcement learning uses a ε-greedy approach as the fundamental policy, where ε is a probability value between [0,1]. Each time an action is selected, there is a probability of ε being exploited in the Q-table and the action with the largest reward is expected, and there is a probability that an action is randomly performed in the exploitation.

3.2. Systematic Action Transitions and Delayed Rewards. Figure 3 demonstrates how the system states change over time. Assuming that at the moment t, the reinforcement learning algorithm, taking DQN as an example, obtains observation o_t from the server state, and makes offloading

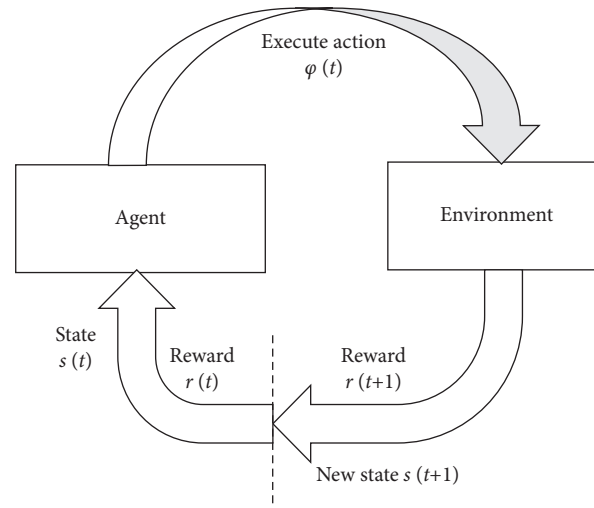

Figure 2: The process of reinforcement learning.

action φ_t based on the observation. Then the offloading action will affect the state of the user who receives the offloading permission, which in turn affects the state of the specific task to be offloaded. Once a task is in the offloading process, it undergoes state transitions such as transmission, arrival at the server cache, execution on the server, and execution completed, which have a continuous impact on the resources and state of the server throughout the transition process.

When the task is executed, the rewards r_t recorded by the system at this moment are returned to the decision-making algorithm for learning. It is clear from the description that at the moment t, the decision-making algorithm does not have access to the immediate rewards for this action but can only obtain the reward for the action at a previous moment. This is a distinctive feature of the incomplete observation system and is a key point for the offloading model to meet the conditions for asynchronous decision-making [25, 26]. Therefore, the cumulative reward of successive decisions constructed by the learning process is the key to determining whether the optimization objective of the system is satisfied.

Assuming that the cumulative positive rewards of reinforcement learning (excluding punitive rewards) are equal to the optimization objective Ψ of the system. Therefore, an upper bound on the cumulative reward is the optimization objective, which can be expressed as

$$\sum_{t \in T} \left| \overline{r}_t + r_t \right| \leq \Psi, \tag{15}$$

where $\overline{r}_t$ is the punitive rewards.

The computation offloading in the edge environment is a complex process of continuous decision-making. And a model-free reinforcement learning method, i.e., temporal-difference (TD), is applied, which combines Monte Carlo sampling and bootstrapping in dynamic programming and usually leads to better learning performance and efficiency. Here, the loss function can be defined as

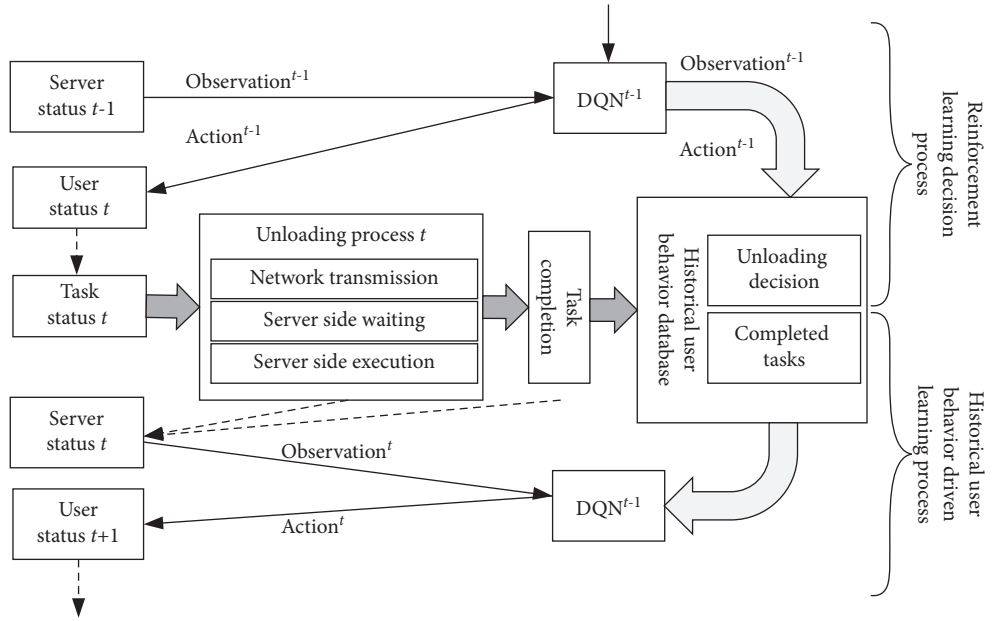

Figure 3: Learning process and delayed reward.

$$L(y, y') = \frac{\sum_{i=1}^{n} (y_i - y_i')^2}{n}, \tag{16}$$

where y and y' are the real value and target value of the model output, respectively.

The proposed learning method runs on an offloading model on the edge server and is mainly as follows:

(1) At the beginning of learning, the server starts the offloading process and updates the reward value $r_t = (T_{n,i}^L/\varepsilon_{n,i})$ of the latest task for that user if the task has been executed

(2) If the system is overloaded, it updates the reward value as $r_t = -|r_{t-1}|$ of the latest task for that user

(3) The server state s_Ω^t, and the list of users φ_Ω^t are obtained and can be offloaded in this decision from the action space. Then the offloading notifications are sent to the users in the list by the server

(4) The server reads the latest reward value of the task and takes the triplet of state, action, and reward as the training input to the reinforcement learning algorithm

3.3. DQN-based Offloading Strategy. To better evaluate the effect of the policies on action selection, the value function about states and actions is converted into a recursive form, which can be denoted as

$$Q(s_t, \varphi_t) = \Phi_t + \zeta \min_{\varphi_{t+1}} Q(s_{t+1}, \varphi_{t+1}), \tag{17}$$

where Φ_t is the value of cost and ζ is the discount coefficient.

In conventional Q-learning algorithms, the number of states in the environment is generally assumed to be relatively small, and therefore a look-up table is used to record the state-action pairs (s, φ). However, since the number of states in the constructed MEC network is so large, it is computationally expensive to update the Q function in an iterative manner if the Q learning algorithm continues to be used. So DQN is proposed to address the problem. The DQN algorithm is a typical value-based policy algorithm that collects the state s_t of the current network environment as input data of the estimated value of the deep network. And the output of the estimated value network is $Q(s_t, \varphi), \forall \varphi \in A$, where Q values corresponds to all the actions. Then, a greedy algorithm, i.e., ε – greedy, is used to select the actions φ_t. Next, the user performs the action φ_t, and the network environment turns to the next state s_{t+1}, while the value of cost Φ_t is generated. Based on this value, the parameters of the estimated value network are updated, and after many iterations of update, the estimated value network has been trained to output the optimal Q function $Q(s, \varphi)$. The mean square error function is used to define the loss function of the estimated value network, which can be written as

$$L_t = E\left[\left(Y_t - Q(s_t, a_t|\theta)\right)^2\right], \tag{18}$$

where θ denotes the weight parameter of the estimated value network, and Y_t represents the value of optimization objective for the estimated value network. However, if an identical deep neural network is used to obtain the target value, the target output of the network also changes with the update of the parameters, that is, the label changes during the deep learning training, which is obviously unreasonable [27]. Therefore, it is necessary to introduce another neural network named as the target value network, whose network structure is exactly the same as the estimated value network. The only difference between them is that the parameter θ of the target value network will not be updated at each timeslot but will be copied from the parameters of the estimated value network after every K steps of training. Namely, the

parameter of the target value network is updated K steps slower than the estimated value network. The target value can be expressed as

$$Y_t = \Phi_t + \zeta \min_{\varphi_{t+1}} Q(s_{t+1}, \varphi_{t+1} | \theta^-). \quad (19)$$

In addition, the samples of training data are independent of each other in supervised learning. But, it is noted that the states of the MEC network are continuous in time series, which affects the reliability of training to some extent. Therefore, an experience replay unit (ERU) is introduced in the DQN network and all the samples coming from the interaction between environments and agents are stored in the memory of the ERU in the form of quaternions $(s_t, \varphi_t, \Phi_t, s_{t+1})$, where s_{t+1} is the next state for the states$_t$. In the training phase, one sample packet is randomly grabbed from the ERU at a time, and the size of the packet can be set arbitrarily within the maximum number of samples so that the temporal correlation between datasets can be broken, making the samples independent and increasing the generalization ability of deep learning. The pseudocode of the DQN-based offloading algorithm is shown in Algorithm 1.

4. Experiments and Analysis

The simulation experiment platform uses MATLAB mathematical software, and the version is 2016a. The computer hardware conditions used in the simulation are as follows: CPU is i7-7200U and the running memory size is 4GB. In the experiments, the simulation scenario is assumed as follows: the bandwidth $B = 5$ MHz, and the computing capacity $F = 12$ GHz/sec, and the computing capacity of each mobile user itself is $f = 5$ GHz/sec. The transmission power of the mobile users is between $p_{\min}$ and $p_{\max}$, where $p_{\min} = 150$ mW and $p_{\max} = 300$ mW. Assuming that the computation offloading follows a uniform distribution between 3000 and 5000 Kb. Besides, decision weights are set as $\omega_t = \omega_e = 0.5$.

4.1. Convergence Procedure in Training DQN Strategy. As shown in Figure 4, three different sets of variables are adopted in the experiments, which are as follows: (1) the number of users is the same as the number of edge servers $(N, M) = (15, 15)$; (2) $(N, M) = (15, 20)$; and (3) $(N, M) = (20, 15)$. And the number of iterations is 20,000.

As shown in Figure 4, in the DQN-based offloading strategy, the average system cost of these three curves decreases rapidly until convergence. When $(N, M) = (15, 15)$, the average cost converges to the lowest value after about 11,000 iterations; when $(N, M) = (15, 20)$, the average cost converges to a stable value after about 9,000 iterations, and this value approximates the value in the case of an equal number of users and edge servers. When $(N, M) = (20, 15)$, the average cost converges to a larger value than that in the other two cases after about 13,000 iterations. Hence, it is confirmed that the proposed strategy allows the system cost to gradually decrease and converge to a stable value regardless of the relationship between the number of users and the number of edge servers.

4.2. Comparison with Other Algorithms. Besides, it is compared with the algorithms in Reference [8], Reference [10], and Reference [15] in the simulation.

4.2.1. Comparison of the Number of Terminals and Average Delay. The relationship between the number of user terminals and the average delay for different algorithms is illustrated in Figure 5.

It can be seen in Figure 5 that since the algorithm proposed in Reference [8] does not consider the collaboration mechanism, the load on the computing server rises with the increase of the number of user terminals, leading to an overall rise in the user task delay. Therefore the average user delay is the biggest. The algorithm proposed in Reference [10] adopts side cloud collaboration, and when the number of user terminals increases, the MEC server can transmit the tasks that cannot be processed in time to the cloud server for execution. Although the side cloud collaboration can reduce the task delay, the average user delay is also relatively high because the cloud server is far away from the user, which increases the transmission delay. Algorithms proposed in this paper and Reference [15] both use reinforcement learning to design the offloading strategy. However, the proposed algorithm constructs a multiuser MEC model to offload computation tasks as quickly as possible or execute them locally, so the delay is the smallest and the average delay does not exceed 0.9 s when the number of terminals is 40.

4.2.2. Relationship between Computing Capacity of MEC Servers and Maximum User Delay. The effect of the computing capacity of MEC servers on the maximum user delay under different algorithms is shown in Figure 6.

From Figure 6, it is illustrated that the user task delay becomes smaller with the increase of the MEC computing capacity. Also, the proposed algorithm has a smaller delay compared to the other three algorithms, and its maximum user delay is about 0.6 s when the computing capacity of MEC servers reaches 16 GHz/sec. This is because when the computing capacity of MEC servers is low, the servers collaborate with each other to balance their load and reduce the task delay. Therefore, the advantages of edge-cloud collaboration will no longer be obvious, and the performance of the algorithm proposed in Reference [8] gradually approaches the curve of Reference [10].

4.2.3. Relationship between Average System Overhead and the Number of Mobile Users. Figure 7 illustrates the performance of the average system overhead with a different number of mobile users, where the number of channels is set to be 12 and the computing capacity of MEC servers is set as $F = 12$ GHz/sec.

As shown in Figure 7, the average system overhead of all four algorithms increases with more and more users. In Reference [10], some of the computation tasks are offloaded to the cloud computing center for execution, and the cloud computing center is far away from users, thereby the TD and

Begin
(1) Emptying the storage area of the ERU
(2) Initialize the weight parameters of the estimated value network θ, and make the parameters of the target value network $\theta^{-} = \theta$
(3) Initialization status s
(4) **For** $t = 1:1:T$
(5) **do**
(6) Under the greedy algorithm, an action is selected based on the state $s_t\varphi_t$
(7) Execute the action φ_t and observe the system costs Φ_t and s_{t+1}
(8) Collecting samples $(s_t, \varphi_t, \Phi_t, s_{t+1})$ and storing them in the ERU
(9) **If the** samples are larger than the size of the sample pack **then** grabbing a sample packet at ERU
(10) **If** s_t is the final state **then** $Y_t = \Phi_t$
Otherwise $Y_t = \Phi_t + \zeta \min_{\varphi_{t+1}} Q(s_{t+1}, \varphi_{t+1}|\theta^{-})$
(11) Execute RMSPropOptimizer, optimize $(Y_t - Q(s, \varphi|\theta))^2$
(12) Every K time slots elapsed, so that $\theta^{-} = \theta$
(13) $s_t = s_{t+1}$
(14) **End**
End

Algorithm 1: DQN-based offloading algorithm.

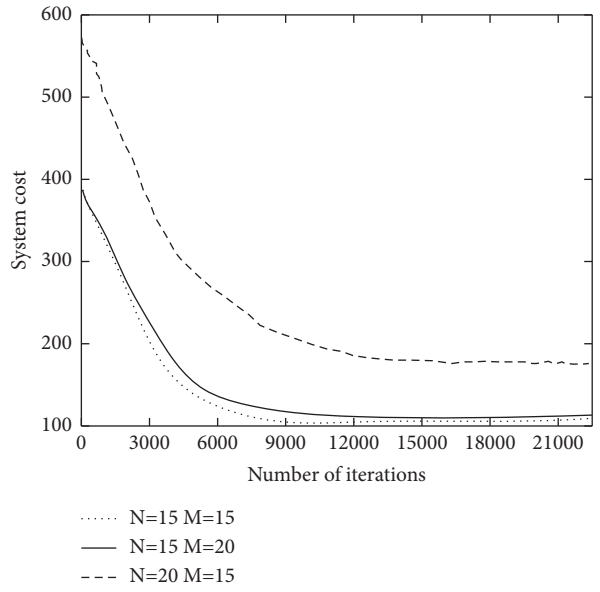

Figure 4: Convergence procedure in training the DQN strategy.

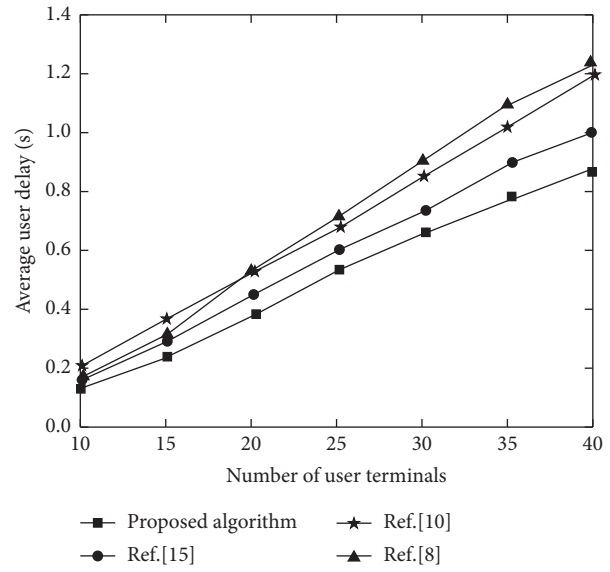

Figure 5: Comparison between the number of terminals and the average delay.

EC increases significantly and its computation overhead is the highest. Heuristic algorithms are applied in Reference [8] for system optimization, while its optimal solution searching performance is weaker compared to the learning results in deep learning networks. Meanwhile, as the number of mobile users increases, the resources that the system can provide in the process of task offloading are limited, so the competition for the limited resources in the system is intense, which can cause an increase in the system delay and energy consumption. In such an environment of intense competition for resources, the metareinforcement algorithm proposed in Reference [15] has a more obvious advantage over the DQN strategy of the proposed algorithm. The proposed algorithm realizes the optimal task unloading and resource allocation through the powerful data optimization ability of DQN. Its total system overhead is less than 120 and can dynamically fine-tune the delay and energy consumption according to the actual needs.

4.2.4. Relationship between Average System Energy Consumption and Training Rounds. When $\omega_t = 0$ and $\omega_e = 1$, the optimization objective can only be concerned about the energy consumption of the whole system and neglect the system delay. Under such circumstances, the performance of energy consumption of the system is shown in Figure 8.

From Figure 8, it can be seen that the average system energy consumption of the four algorithms tends to be stable as the number of training rounds increases, but the proposed algorithm has the smallest average system energy consumption, which tends to be 65 J when the number of training rounds exceeds 19. The proposed algorithm uses

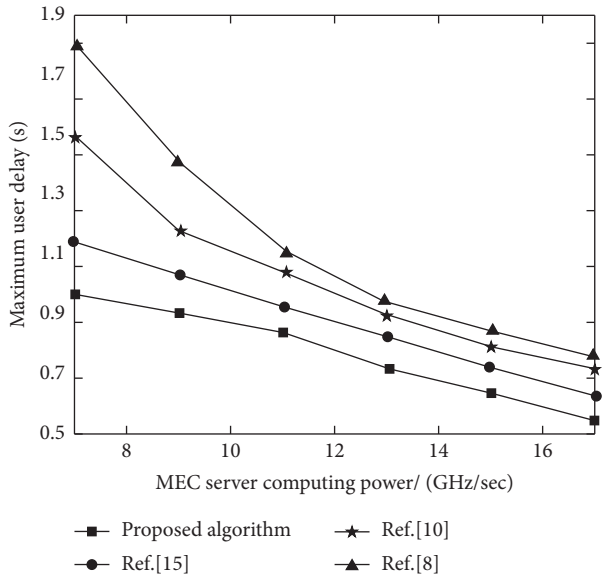

Figure 6: Relationship between computing capacity of MEC servers and maximum user delay.

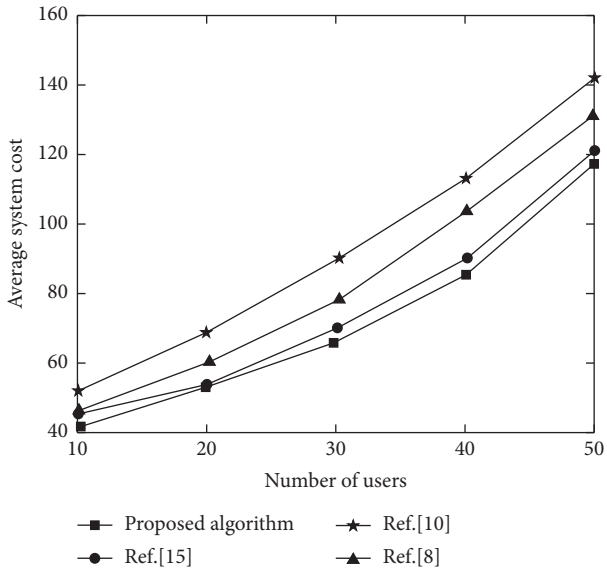

Figure 7: Relationship between average system overhead and the number of mobile users.

DQN to construct an offloading strategy in which the system optimization search is accelerated by system action transition with delayed reward and introduces the MEC system model, leading to smaller overall energy consumption. The meta-reinforcement strategy proposed in Reference [15] is computationally complex and lacks a reasonable system architecture, so the energy consumption increases. In Reference [10], the task offloading is executed based on the cloud edge collaborative architecture, but its stable energy consumption is more than 100 J due to the long distance between cloud computing centers and the users. Reference [8] has less computational overhead due to low complexity.

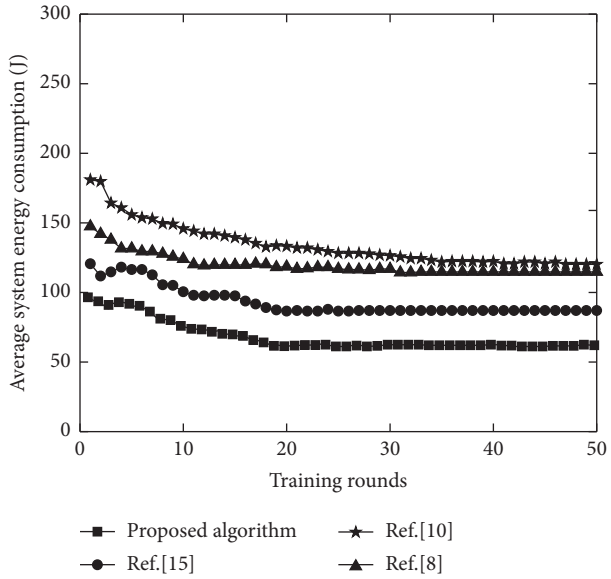

Figure 8: Relationship between average system energy consumption and training rounds.

However, the time delay is large, so the overall offloading is not effective.

5. Conclusion

To alleviate the network load and reduce the risk of network congestion in traditional cloud computing of IoT, MEC is introduced to formulate the multiuser computing offloading problem, where the optimization objective is set to minimize the total weighted overhead of time delay and energy consumption to ensure reasonable system resource allocation. Additionally, the optimization problem is solved using a DQN-based offloading strategy, obtaining the optimal scheme. The results based on the simulation platform show that:

(1) The introduction of MEC, which enables the computation task in close proximity to the users, can reduce system time and energy consumption. The average delay of the proposed algorithm does not exceed 0.9 s, and the average energy consumption tends to be 65 J when the number of user terminals is 40.

(2) The proposed algorithm can execute computational tasks as close as possible by adopting the MEC system, enabling it to reduce the total system overhead to a great extent. And the total overhead is lower than 120 when the number of users is 50.

As the mobile user devices have very limited resources and suffer from the problem of battery aging, the energy provided to the users is not enough for them to complete the whole offloading procedure in some cases. Therefore, how to renew the energy for mobile users to ensure that offloading will not be interrupted deserves deep study, which will also be the focus of our research in the future.

Data Availability

The data included in this paper are available from the author upon request without any restriction.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] J. Guo and N. Shah, "Internet of things based intelligent techniques in workable computing: an overview," *Scientific Programming*, vol. 2021, Article ID 6805104, 15 pages, 2021.

[2] X. Wu, J. Gan, S. Chen, X. Zhao, and Y. Wu, "Optimization strategy of task offloading with wireless and computing resource management in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 8288836, 11 pages, 2021.

[3] K. Wang, Z. Hu, Q. Ai et al., "Joint offloading and charge cost minimization in mobile edge computing," *IEEE Open Journal of the Communications Society*, vol. 1, no. 4, pp. 205–216, 2020.

[4] W. Zhan, C. Luo, and G. Min, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 99, pp. 3341–3356, 2020.

[5] Y. Fu, X. Yang, and P. Yang, "Energy-efficient offloading and resource allocation for mobile edge computing enabled mission-critical internet-of-things systems," *EURASIP Journal on Wireless Communications and Networking*, vol. 10, no. 1, pp. 1–16, 2021.

[6] U. Saleem, Y. Liu, and S. Jangsher, "Latency minimization for d2d-enabled partial computation offloading in mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 99, pp. 4472–4486, 2020.

[7] Z. Kuang, L. Li, and J. Gao, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 9, pp. 6774–6785, 2019.

[8] L. I. Shulei, D. Zhai, and P. Du, "Energy-efficient task offloading, load balancing, and resource allocation in mobile edge computing enabled IoT networks," *Science China Information Sciences*, vol. 62, no. 2, pp. 1–3, 2019.

[9] X. Chen, Z. Liu, Y. Chen, and Z. Li, "Mobile edge computing based task offloading and resource allocation in 5G ultra-dense networks," *IEEE Access*, vol. 7, no. 5, pp. 184172–184182, 2019.

[10] C. Jiang, Y. Li, and J. Su, "Research on new edge computing network architecture and task offloading strategy for Internet of Things," *Wireless Networks*, vol. 5, no. 2, pp. 1–13, 2021.

[11] Q. Wang, S. Guo, J. Liu, and Y. Yang, "Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing," *Sustainable Computing: Informatics and Systems*, vol. 21, no. 3, pp. 154–164, 2019.

[12] L. Liu, X. Qin, Z. Zhang, and P. Zhang, "Joint task offloading and resource allocation for obtaining fresh status updates in multi-device MEC systems," *IEEE Access*, vol. 8, no. 5, pp. 38248–38261, 2020.

[13] Z. Zhang, C. Li, and S. L. Peng, "A new task offloading algorithm in edge computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2, no. 1, pp. 1–21, 2021.

[14] C. Li, M. Song, H. Tang, and Y. Luo, "Offloading and system resource allocation optimization in TDMA based wireless powered mobile edge computing," *Journal of Systems Architecture*, vol. 98, no. 7, pp. 221–230, 2019.

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

[16] H. Lu, C. Gu, F. Luo, W. Ding, S. Zheng, and Y. Shen, "Optimization of task offloading strategy for mobile edge computing based on multi-agent deep reinforcement learning," *IEEE Access*, vol. 8, no. 7, pp. 202573–202584, 2020.

[17] K. Wang, X. Wang, X. Liu, and A. Jolfaei, "Task offloading strategy based on reinforcement learning computing in edge computing architecture of internet of vehicles," *IEEE Access*, vol. 8, no. 9, pp. 173779–173789, 2020.

[18] C. Li, W. Chen, J. Tang, and Y. Luo, "Radio and computing resource allocation with energy harvesting devices in mobile edge computing environment," *Computer Communications*, vol. 145, no. 9, pp. 193–202, 2019.

[19] P. Paymard, S. Rezvani, and N. Mokari, "Joint task scheduling and uplink/downlink radio resource allocation in PD-NOMA based mobile edge computing networks," *Physical Communication*, vol. 32, no. 1, pp. 160–171, 2019.

[20] Y. He, L. Ma, R. Zhou, C. Huang, and Z. Li, "Online task allocation in mobile cloud computing with budget constraints," *Computer Networks*, vol. 151, no. 3, pp. 42–51, 2019.

[21] F. Wang, J. Xu, and S. Cui, "Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2443–2459, 2020.

[22] Y. Li and C. Jiang, "Distributed task offloading strategy to low load base stations in mobile edge computing environment," *Computer Communications*, vol. 164, no. 8, pp. 240–248, 2020.

[23] K. Wang, Z. Xiong, L. Chen, P. Zhou, and H. Shin, "Joint time delay and energy optimization with intelligent overclocking in edge computing," *Science China (Information Sciences)*, vol. 63, no. 4, pp. 154–169, 2020.

[24] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, no. 7, pp. 26652–26664, 2019.

[25] Z. Wang, P. Li, S. Shen, and K. Yang, "Task offloading scheduling in mobile edge computing networks," *Procedia Computer Science*, vol. 184, no. 4, pp. 322–329, 2021.

[26] S. Hu and G. Li, "Dynamic request scheduling optimization in mobile edge computing for IoT applications," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1426–1437, 2020.

[27] H. Guo, J. Zhang, J. Liu, and H. Zhang, "Energy-aware computation offloading and transmit power allocation in ultradense IoT networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4317–4329, 2019.