future internet

MDPI

Article

A Task Offloading and Resource Allocation Strategy Based on Multi-Agent Reinforcement Learning in Mobile Edge Computing

Guiwen Jiang [1], Rongxi Huang [2,*], Zhiming Bao [1] and Gaocai Wang [3]

[1] School of Artificial Intelligence Technology, Guangxi Technological College of Machinery and Electricity, Nanning 530007, China; jgw998311038@163.com (G.J.); 13077796550@163.com (Z.B.)
[2] School of Information Engineering, Guangxi Vocational University of Agriculture, Nanning 530007, China
[3] School of Computer and Electronic Information, Guangxi University, Nanning 530004, China; wanggcgx@163.com
* Correspondence: huangrx@gxnzd.edu.cn

Abstract: Task offloading and resource allocation is a research hotspot in cloud-edge collaborative computing. Many existing pieces of research adopted single-agent reinforcement learning to solve this problem, which has some defects such as low robustness, large decision space, and ignoring delayed rewards. In view of the above deficiencies, this paper constructs a cloud-edge collaborative computing model, and related task queue, delay, and energy consumption model, and gives joint optimization problem modeling for task offloading and resource allocation with multiple constraints. Then, in order to solve the joint optimization problem, this paper designs a decentralized offloading and scheduling scheme based on "task-oriented" multi-agent reinforcement learning. In this scheme, we present information synchronization protocols and offloading scheduling rules and use edge servers as agents to construct a multi-agent system based on the Actor–Critic framework. In order to solve delayed rewards, this paper models the offloading and scheduling problem as a "task-oriented" Markov decision process. This process abandons the commonly used equidistant time slot model but uses dynamic and parallel slots in the step of task processing time. Finally, an offloading decision algorithm TOMAC-PPO is proposed. The algorithm applies the proximal policy optimization to the multi-agent system and combines the Transformer neural network model to realize the memory and prediction of network state information. Experimental results show that this algorithm has better convergence speed and can effectively reduce the service cost, energy consumption, and task drop rate under high load and high failure rates. For example, the proposed TOMAC-PPO can reduce the average cost by from 19.4% to 66.6% compared to other offloading schemes under the same network load. In addition, the drop rate of some baseline algorithms with 50 users can achieve 62.5% for critical tasks, while the proposed TOMAC-PPO only has 5.5%.

Keywords: mobile edge computing; computing offloading; resource allocation; multi-agent

check for updates

Citation: Jiang, G.; Huang, R.; Bao, Z.; Wang, G. A Task Offloading and Resource Allocation Strategy Based on Multi-Agent Reinforcement Learning in Mobile Edge Computing. *Future Internet* **2024**, *16*, 333. https://doi.org/10.3390/fi16090333

Academic Editors: Armir Bujari, Gabriele Elia and Johann M. Marquez-Barja

Received: 15 August 2024
Revised: 3 September 2024
Accepted: 9 September 2024
Published: 11 September 2024

1. Introduction

Research shows that the number of Internet of Things (IoT) devices is expected to reach 30.9 billion in 2025, and the amount of data generated will exceed 175 zebytes [1,2]. The number of mobile devices is growing exponentially, and the communication and computing capabilities of devices are facing serious challenges [3,4]. As one of the solutions, cloud computing usually causes high delay and privacy leakage due to the long distance of deployment [5,6]. Mobile edge computing (MEC) aims to make up for the shortcomings of cloud computing, decentralize computing resources to the network edge user side, and provide mobile users with short-distance and low-delay computing services. Although computing offloading has been a research hotspot in the field of MEC, the majority of existing schemes have the following problems: (1) Many studies have failed to consider the actual situation, ignoring the possibility of terminal mobility management and network failure, and also failed to consider the information observation and synchronization of network equipment. (2) Some studies have used reinforcement learning (RL) technology to solve the offloading decision problem, but do not consider the training difficulties that may be caused by the delayed reward feature of the MEC environment [7]. (3) Although some studies have taken into account the distributed characteristics of MEC, they have failed to decompose the decision problem. Instead, they choose single-agent RL to solve the decision problem of the whole system, which may lead to problems such as too large a dimension of decision space or difficult convergence of training.

Given the uncertainties faced in real network environments such as system failures and user mobility for crossing regions, how to ensure reliable communication and collaboration among MEC network devices, and how to handle efficiently task offloading and resource allocation on edge servers, are all challenges currently faced in the MEC field. Therefore, this paper proposes a distributed task offloading and resource allocation scheme based on "task-oriented" multi-agent reinforcement learning, which can effectively reduce system average cost, delay, energy consumption, and task drop rate. The main contributions of this paper are as follows:

(1) This paper constructs a cloud-edge collaborative computing model, and related task queue, delay, and energy consumption model, and gives joint optimization problem modeling for task offloading and resource allocation with multiple constraints.
(2) In order to solve the joint optimization problem, this paper designs a decentralized task offloading and resource allocation scheme based on "task-oriented" multi-agent reinforcement learning. In this scheme, we present information synchronization protocol and offloading scheduling rules and use edge servers as agents to construct a multi-agent system based on the Actor–Critic framework.
(3) An offloading decision algorithm TOMAC-PPO (Task-Oriented Multi-Agent Collaborative-Proximal Policy Optimization) is proposed. The algorithm applies the proximal policy optimization to the multi-agent system and combines the Transformer neural network model to realize the memory and prediction of network state information. Experimental results show that this algorithm has better convergence speed, and can effectively reduce the service cost, energy consumption, and task drop rate under high load and high failure rates.

The structure of this paper is as follows. Section 2 introduces the related works; Section 3 establishes a cloud-edge collaborative model; Section 4 proposes a distributed task offloading and resource allocation scheme based on task-oriented multi-agent reinforcement learning; Section 5 verifies the performance of the algorithm proposed in this paper; Section 6 summarizes this paper and gives future research directions.

2. Related Works

Many scholars have conducted extensive research on optimizing the task offloading scheduling scheme of MEC networks to improve network service quality. Traditional offloading optimization schemes usually establish mathematical programming models to solve optimization problems. For example, the authors established a 0-1 integer programming model for joint optimization of energy consumption and delay in [8], and designed a service request distribution method based on game theory in [9]. However, such schemes ignore the mobility of devices and information observation, making it difficult to adapt to dynamic and highly random large-scale network environments. The authors used a filling method based on linear programming to solve resource allocation problems and conduct joint optimization with dynamic pricing problems in [10]. The authors proposed an asynchronous computing framework, and the general benders decomposition method is used to decompose and iteratively solve the problem of user scheduling and resource allocation in [11]. These traditional optimization schemes typically have high operational efficiency and a solid theoretical foundation but often require accurate system state information, which is challenging in practical MEC systems. Moreover, traditional optimization schemes often struggle to cope with the high-dimensional decision space in optimization problems due to their high computational complexity.

Artificial intelligence solutions utilizing neural networks have been proven to be effective for complex task offloading and resource allocation problems. RL techniques are often used to solve offloading decision problems. Due to the fundamental idea of RL being to enable agents to interact with their environment and learn the optimal strategy to achieve their goals, RL is suitable for solving offloading decision problems.

In recent years, traditional RL methods have gradually been replaced by deep reinforcement learning (DRL), such as the DRL method used to solve offloading decisions, while also addressing the allocation of bandwidth, cache, and computing power in [12]. The authors combined DRL to solve the optimal offloading strategy and federated learning methods to address data privacy issues in [13]. Compared with the previous optimization methods, this type of scheme does not require the establishment of a mathematical model for solving and can autonomously explore the optimal offloading strategy without prior knowledge, making it more suitable for high-dimensional state spaces in large-scale networks. However, these studies typically require a significant amount of time and data to train neural networks, and all adopt centralized decision architectures, resulting in a high dependence on the central control node, leading to low robustness, poor scalability, and other issues. In addition, DRL has been applied to unmanned aerial vehicles (UAVs) for navigation, trajectory planning, and radio resource management [14,15].

The performance differences between traditional Q-learning and deep Q network (DQN) algorithms in solving task offloading scheduling problems were compared in [16]. However, such schemes do not consider distributed architecture, which reduces robustness while increasing training difficulty. Therefore, the authors adopt the asynchronous advantage Actor–Critic algorithm to achieve a distributed architecture, which accelerates training while reducing the correlation between state transition samples in [17]. However, unlike parallel RL architectures such as A3C, all nodes in MEC networks are not in independent environments that do not affect each other. Therefore, distributed offloading decision-making is a multi-agent problem, and using traditional single-agent or parallel RL schemes may make it difficult for the algorithm to converge [18]. Furthermore, some scholars have also begun to use multi-agent reinforcement learning (MARL) technology to solve task-offloading decision problems. For example, reference [19] proposed a multi-agent DQN algorithm based on value decomposition for task offloading strategy problems. The authors used the multi-agent deep determining policy Gradient (MADDPG) algorithm to solve the multi-agent task offloading problem [20]. However, existing research on DRL-based offloading decisions has not taken into account the training difficulties that may arise from reward delay in MEC network environments.

3. Preliminaries: Network Model and Problem Definition

3.1. Cloud-Edge Collaboration Model for MEC Network

As shown in Figure 1, the cloud-edge collaboration model can be divided into user layer, edge layer and cloud layer. The user layer is composed of m mobile devices used by all users in the service area, and it can be denoted by $UD = \{ud_1, ud_2, \cdots ud_m\}$. User equipment may move locations and generate computing tasks at any time. The edge layer includes n edge nodes, which are represented as $EN = \{en_1, en_2, \cdots en_n\}$. Each node consists of a wireless base station and an edge server.

3.2. Task Queue Description

The queue model of this paper is shown in Figure 2. Each user device maintains a cache, a computing queue, and a transmission queue. After a task is generated, it first enters the cache to wait for an offloading decision. After the offloading decision is determined, the task can enter the calculation queue to perform local computing or enter the transmission queue to offload to edge layer devices. The ud_i lengths of the cache area, computing queue, and transmission queue recorded are q_i^{comp}, q_i^{cache}, and q_i^{tran}, respectively. The available cache capacity of the user's device is R^{ud}.

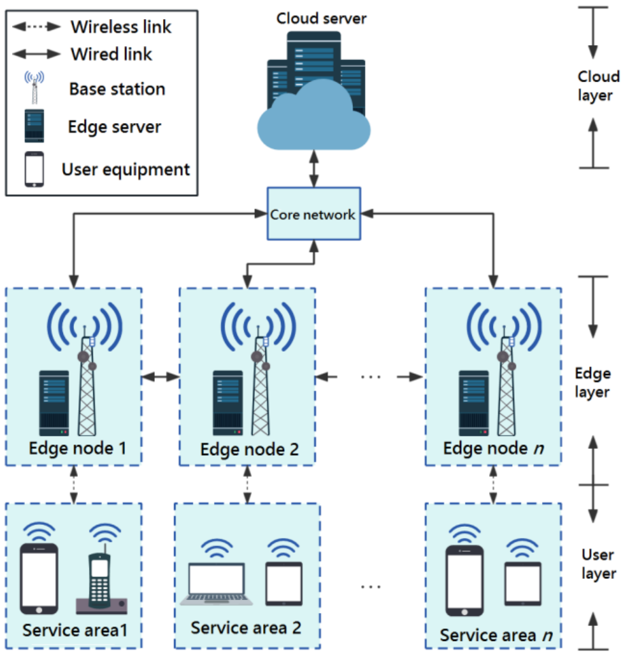


Figure 1. Cloud-edge collaboration model for MEC network.

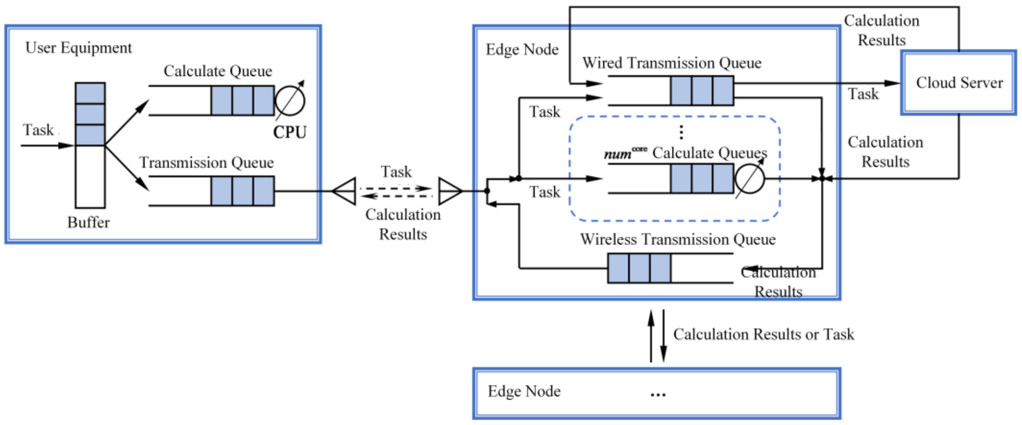


Figure 2. Task computing and transmission queues.

Each edge server maintains a wireless transmission queue, a wired transmission queue, and num^{core} (the number of CPU cores in the edge server) computing queue. If a task is offloaded to the edge server, it can enter the server's computing queue or enter the server's wired transmission queue to be sent to other nodes or the cloud. If the task is completed within the edge server or cloud server, the computing result can be sent back to the user in the wireless transmission queue of the edge server. The en_j length of the computing queue and transmission queue, as well as the cache capacity of a single edge server, are R^{en}, $q_j^{\text{en-tran}}$, and $q_j^{\text{en-comp}}$, respectively, and all queues comply with FIFO (First In First Out).

Due to the different priorities of various tasks, a simple FIFO queue is difficult to meet practical needs. Therefore, this paper allows computing devices to perform queue insertion arrangements when receiving tasks. When the decision node makes an offload decision on task x_t^i, the priority weight $qf_{i,t}$ of the task can be determined. Task x_t^i is queued according to weight $qf_{i,t}$ when it enters any queue, and, the higher the weight, the higher the priority.

To further solve queue congestion and improve throughput and delay performance, this paper also introduces an active queue management mechanism, which is described as follows:

1. If the average queue length is less than the minimum threshold th_{min}, newly arrived tasks are pushed into the queue;
2. If the average queue length is greater than the minimum threshold th_{min} and less than the maximum threshold th_{max}, randomly drop the newly arrived task according to probability;
3. If the queue length has reached the maximum threshold th_{max}, the newly arrived task is dropped p_{drop}.

3.3. Task Model

Let x_t^i denote the generated task at the time t of the mobile device ud_i, and its deadline is $\tau_{i,t}$. The 0-1 variable $D_{i,t}^k(0 \le k \le n+2)$ indicates the offloading status of the task x_t^i, and we have the following.

$$D_{i,t}^k = \begin{cases} 1\ , \text{ For } x_t^i, \text{ the offloading goal is } en_k \\ 0\ , \text{ the offloading goal is not } en_k \end{cases} ,1 \le k \le n \tag{1}$$

Here, if $D_{i,t}^0$ is 1, the task x_t^i will be performed in local server. If $D_{i,t}^{n+1}$ is 1, x_t^i will be offloaded at edge sever. If $D_{i,t}^{n+2}$ is 1, x_t^i will be discarded by the user. The offloading decision vector $\boldsymbol{D}_{i,t} = \left(D_{i,t}^0, D_{i,t}^1, \cdots D_{i,t}^{n+2}\right)$ can uniquely determine the offloading direction of the task x_t^i. When the task x_t^i is offloaded to a node en_j, the weight of CPU computing resources allocated to the task x_t^i is expressed as $ff_{i,t}^j$. When the server is running, it dynamically allocates its computing resources to each task according to its weight. Specifically, the computing power $ff_{i,t}^j$ that can be occupied by the task x_t^i executed on nodes can be expressed as follows:

$$f_{i,t}^j = \frac{ff_{i,t}^j}{ff_{\text{sum}}^j} \cdot f^{\text{edge}} \tag{2}$$

where f^{edge} represents the computing power owned by a single edge node and ff_{sum}^j represents the total computing power weight of all tasks performing computing in the node en_j.

This paper categorizes computing tasks into four categories based on their requirements for security and delay: high-priority tasks, critical tasks, low-priority tasks, and regular tasks. The specific classification description is as follows:

(1) High-priority tasks have high requirements for delay, which are related to security and have hard indicators for task completion rate.
(2) The key task is a computing task that requires extremely high security and can appropriately lower delay requirements but cannot be discarded.
(3) Low-priority tasks are not related to security but are in scenarios that require energy savings, or tasks with excessive data volume and computation time.
(4) Except for the above three types of computing tasks, all other computing tasks are routine tasks.

3.4. Network Communication and Observation Model

Suppose the radio channel bandwidth resource of a single base station is B, and the service radius is r. In addition, we assume that the base station has dynamic channel allocation capability, the user ud_i allocated bandwidth weight for the node en_j is $bf_{i,j}$, and then the allocated bandwidth $B_{i,j}$ can be expressed as follows:

$$B_{i,j} = \frac{bf_{i,j}}{\sum_{x=1}^m bf_{x,j}} \cdot B \tag{3}$$

Here, if the user ud_i leaves the service area, the $bf_{i,j} = 0$. The above Formula (3) represents the proportion of the bandwidth allocated by the base station to a user in the total bandwidth of the base station. So, the maximum data transmission rate between ud_i and en_j can be expressed as follows:

$$C_{i,j} = B_{i,j} log_2\left(1 + \frac{PG_{i,j}}{\sigma^2}\right) \tag{4}$$

where P is the transmission power of the sender's equipment. $G_{i,j}$ is the channel gain between ud_i and en_j. σ^2 is the Gaussian noise power of the channel.

Electromagnetic waves travel at the speed of light V_{space} in the service area. For wired communication, the link transmission rate is C_{fiber} and the link propagation rate is V_{fiber}. Each line has a fault probability p_{fiber} in every minute, and communication capability can be recovered after maintenance time T_{repair}.

The MEC network model considered in this paper also has the problem of local observation. A single device only has the ability of local observation, and the global state information of the network can only be obtained by synchronous communication between devices. In addition, for user equipment, the computing density and computing result size of the task cannot be estimated.

3.5. Network Delay and Energy Consumption Model

The task x_t^i generated by the node ud_i is performed on any device at any time t, the delay can be expressed as follows:

$$T_{i,t}^{\text{comp}} = \frac{\lambda_{i,t}\rho_{i,t}}{f} \tag{5}$$

where $\lambda_{i,t}$ is the amount of input data for x_t^i, and $\rho_{i,t}$ is the computing density of x_t^i. That is the number of CPU cycles required to process each bit of data for x_t^i. f is the CPU frequency of the computing device.

The sum of transmission delay and propagation delay generated by communication between ud_i and en_j can be expressed as follows:

$$T_{i,j}^{\text{space}} = \frac{\lambda}{C_{i,j}} + \frac{d_{i,j}}{V_{\text{space}}} \tag{6}$$

where λ is the size of the sent data and $d_{i,j}$ is the distance between ud_i and en_j.

The sum of transmission delay and propagation delay generated by communication between adjacent edge nodes can be expressed as follows:

$$T_{\text{fiber}} = \frac{\lambda}{C_{\text{fiber}}} + \frac{d_{\text{edge}}}{V_{\text{fiber}}} \tag{7}$$

where d_{edge} is the distance between adjacent nodes.

The delay caused by edge node communication with cloud server can be expressed as follows:

$$T_{\text{cloud}} = \frac{\lambda}{C_{\text{fiber}}} + \frac{d_{\text{cloud}}}{V_{\text{fiber}}} + T_{\text{delay}}^{\text{cloud}} \tag{8}$$

where d_{cloud} is the distance between edge nodes and cloud server, and $T_{\text{delay}}^{\text{cloud}}$ is the forwarding delay of core network.

Local calculated energy consumption for x_t^i can be expressed as follows:

$$E_{i,t}^{\text{comp}} = \kappa\lambda_{i,t}\rho_{i,t}(f_i^{\text{user}})^2 \tag{9}$$

where κ is the energy consumption efficiency coefficient of user equipment. f_i^{user} is the CPU frequency of ud_i. If data are sent from ud_i to en_j, the duration $T_{i,j}^{\text{space}}$ can be calculated by (6), and the energy consumption of user equipment during the period can be expressed as follows:

$$E_{i,j}^{\text{tran}} = P_i T_{i,j}^{\text{space}} \tag{10}$$

where P_i is the transmission power of ud_i.

3.6. Offloading Decision and Resource Allocation Problem Modeling

The cost of computing task in this paper can be expressed as follows:

$$Z\left(x_t^i\right) = \varphi_1 \frac{T_{i,t}^{\text{total}}}{\tau_{i,t}} + \varphi_2 \frac{E_{i,t}^{\text{total}}}{\mu_1 \lambda_{i,t} \rho_{i,t}} + drop_{i,t} \tag{11}$$

where $T_{i,t}^{\text{total}}$ is the total processing delay of task x_t^i and $E_{i,t}^{\text{total}}$ is the total energy consumption of x_t^i. $\rho_{i,t}$ is the calculated density of x_t^i. $\lambda_{i,t}$ is the amount of input data for x_t^i. $\tau_{i,t}$ is the upper limit of the tolerable delay of x_t^i. The constant μ_1 is the scaling factor, so that $0 < \frac{E_{i,t}^{\text{total}}}{\mu_1 \lambda_{i,t} \rho_{i,t}} \leq 1$. In order to achieve the effect of normalization, variable $drop_{i,t}$ is defined as follows:

$$drop_{i,t} = \begin{cases} 1\ ,\ x_t^i \ \textit{is discarded} \\ 0\ ,\ x_t^i \ \textit{completes the calculation} \end{cases} \tag{12}$$

For each task in this paper, φ is set as follows. The optimization target for high-priority tasks does not include energy consumption, so there is $\varphi_2 = 0$ for high-priority tasks. The optimization target for critical mission only considers the drop rate, so there is $\varphi_1 = \varphi_2 = 0$ for critical missions. The optimization goal of low-priority tasks does not include delay, so there is $\varphi_1 = 0$ for low-priority tasks. The optimization of routine tasks needs to comprehensively consider the delay, energy consumption, and drop rate, so there is $\varphi_1 = \varphi_2 = 1$ for routine tasks.

The goal of this paper is to optimize the task offloading decision of each decision-making device and the bandwidth allocation decision of the server to the user equipment, so as to minimize the sum of all system task costs in a long period of time. Thereby reducing network delay, drop rate and user energy consumption. The above optimization problem can be modeled as follows:

$$\begin{aligned}
&\min_{\boldsymbol{D,ff,qf,bf}} \left[\sum_{1 \le i \le m} \sum_{t \in T_i^{\text{task}}} Z\left(x_t^i\right) \right] \\
&\text{s. t. C1} : D_{i,t}^k \in \{0,1\}\ ,\forall i \le m,\ \forall t \in T_i^{\text{task}} \\
&\text{C2} : \sum_{k=0}^{n+2} D_{i,t}^k = 1\ ,\ \forall i \le m,\ \forall t \in T_i^{\text{task}} \\
&\text{C3} : ff_{i,t}^j \in [0,1], x_t^i \text{ is calculated in } en_j,\ \forall i \le m,\ \forall t \in T_i^{\text{task}} \\
&\text{C4} : qf_{i,t} \in [0,1],\ \forall i \le m,\ \forall t \in T_i^{\text{task}} \\
&\text{C5} : bf_{i,j} \in [0,1],\ \textit{communication} \text{ between } ud_i \text{ and } en_j,\ \forall i \le m,\ \forall j \le n \\
&\text{C6} : q_i^{\text{cache}} + q_i^{\text{comp}} + q_i^{\text{tran}} \le R_i^{\text{ud}}\ ,\ \forall i \le m \\
&\text{C7} : q_j^{\text{en-comp}} + q_j^{\text{en-tran}} \le R^{\text{en}}\ ,\ \forall j \le n \\
&\text{C8} : T_{i,t}^{\text{total}} \le \tau_{i,t}\ ,\ \forall i \le m,\ \forall t \in T_i^{\text{task}} \\
&\text{C9} : d_{i,j} \le r\ ,\ \textit{communication} \text{ between } ud_i \text{ and } en_j,\ \forall i \le m,\ \forall j \le n
\end{aligned} \tag{13}$$

where D is the set of offloading decision vectors $D_{i,t}$ corresponding to all tasks. ff is the set of weights $ff_{i,t}^j$ assigned to the computing power corresponding to all tasks. qf is the set of queue priority weights corresponding to all tasks. bf is the set of bandwidth allocation weights $bf_{i,j}$ for all users. $T_{i,t}^{\text{total}}$ is the total delay for task processing. Setting T_i^{task} contains all the moments that generate tasks within the system runtime. Constraint C1 indicates that each task has only two states for offloading the target device: completing offloading and no offloading. Constraint C2 indicates that each task has and only has one offloading decision result. Constraints C6 and C7 indicate that the sum of the queue and cache lengths of user devices or edge nodes does not exceed their cache capacity. Constraint C8 indicates that the total processing delay of the task does not exceed its upper of tolerance delay. Constraint C9 indicates that the distance between the user and the edge node during communication does not exceed the service radius of the node base station.

4. A Joint Optimization Scheme for Task-Oriented Multi-Agent PPO Offloading Decision and Resource Allocation

4.1. Network Information Synchronization Protocol

At the user layer, users periodically report their own location coordinates to the edge nodes. After receiving the information, the node regards itself connected to the user and records the location information. At the edge layer, each edge node sends synchronous data to all adjacent nodes, so the node can obtain the topology of the edge layer and construct a routing table after receiving the data and store the status information of the remaining nodes in its own database. In the cloud layer, cloud server periodically sends synchronization information to all edge nodes to confirm link connectivity. As shown in Figure 3, all synchronization information is sent periodically. If synchronization is not received after a period of timeout, the node considers that it has disconnected from the device and does not retransmit.

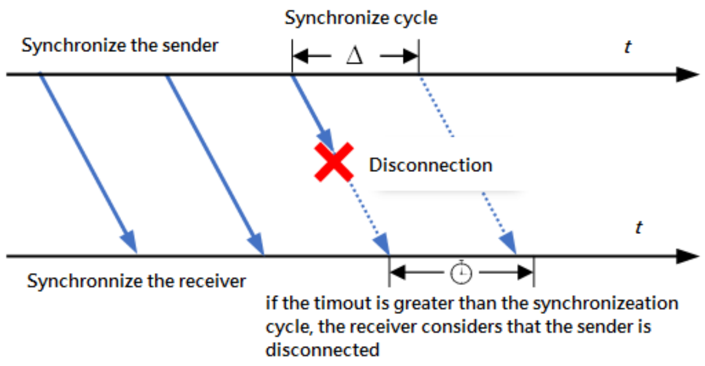

Synchronize cycle
Synchronize the sender
Δ
t
Disconnection
t
Synchronnize the receiver
if the timout is greater than the synchronizeation cycle, the receiver considers that the sender is disconnected

Figure 3. Synchronization timeout due to disconnection (The blue arrow is the direction of information transmission, and the red cross indicates that the sender and receiver are not connected).

4.2. Distributed Offloading Scheduling Rules

Considering the mobility of users in MEC network and the hidden trouble of nodes and lines, it is necessary to design a set of offloading scheduling rules to deal with a variety of unexpected situations. In an ideal situation, the offloading scheduling rules are shown in Figure 4.

The description is as follows:

(1) If the user is within the service area, when they generate a task, they first report the summary information of the task to the nearest edge node they are connected to (referred to as the "decision node").
(2) The decision node makes an offloading decision based on this summary information and network conditions and sends it back to the user.
(3) Users perform local calculations or offload tasks to decision nodes based on the received decision results.
(4) After receiving task data, the node forwards the task to the offloading target node for calculation.
(5) After the calculation is completed, offloading the target node will determine the node closest to the user as the "return node", and then send the calculation result to the return node in the shortest path.
(6) The final calculation result is sent back to the user by the return node.

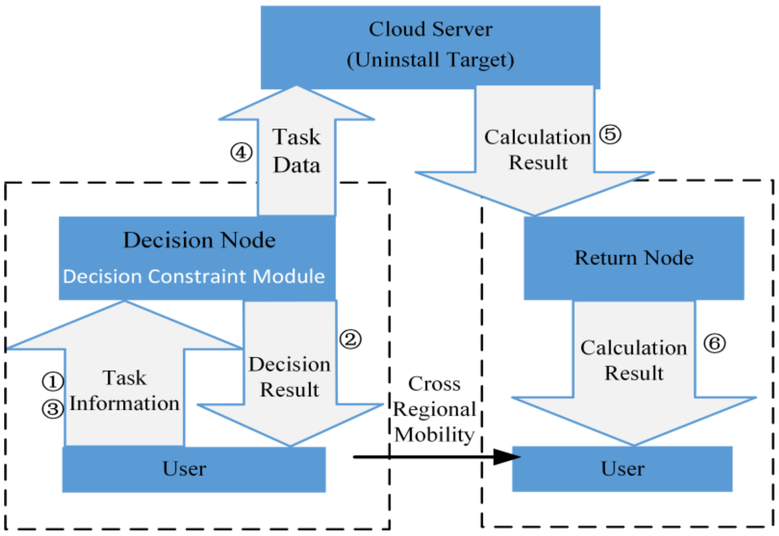


Figure 4. Offloading and scheduling when user moves across cell (The numbers ①–⑥ represent the order in which the scheduling rules are executed).

4.3. Multi-Agent System Based on Actor–Critic Framework

This section proposes a multi-agent system architecture suitable for MEC networks based on the classic strategy learning framework Actor–Critic. The multi-agent system that was designed is shown in Figure 5.

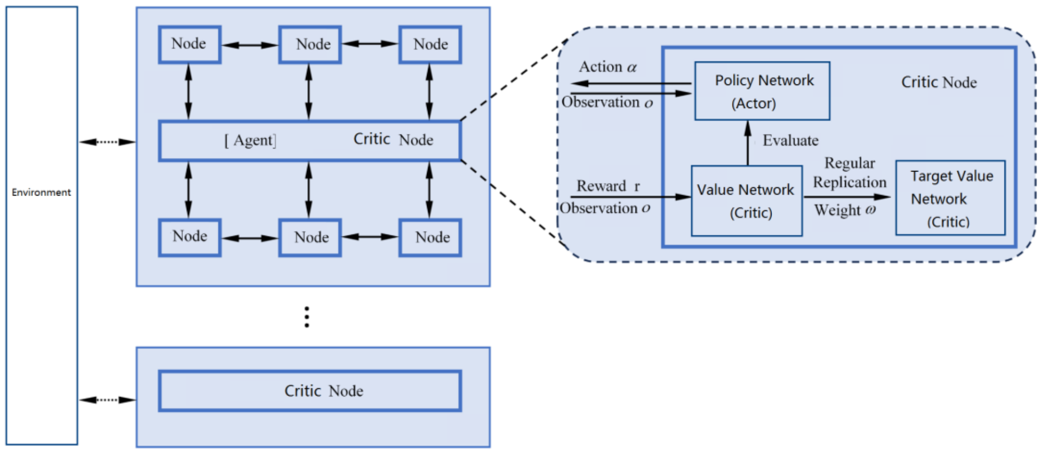


Figure 5. Multi-agent system based on the Actor–Critic framework.

The system adopts the region partitioning method, dividing the entire network into multiple subnets based on the hop distance. If an undirected graph is used to represent a node network, the system will select the node with the highest degree as the commentator node for each subnet in the network graph every time the network topology changes. The commentator node will act as the sole agent in the subnet, making decisions for all nodes within the subnet. Ordinary nodes only need to submit their observed information to the commentator node as a decision basis and execute the action values issued by the commentator node. The commentator node maintains multiple policy networks and a value network internally, with each policy network corresponding to the offloading strategy of each node in the subnet. The intelligent agent can make decisions and execute actions based on the output values of the policy network. Subsequently, the intelligent agent will use the value network to evaluate this action based on environmental feedback rewards and new observation information. The policy network and value network will update their parameters accordingly.

4.4. Task-Oriented Markov Decision-Making Process

This paper proposes a task-oriented MDP model for MEC networks, which has an indefinite length of time steps and multiple time steps are performed simultaneously. As shown in Figure 6, the $x^i_{t_1}$ time step corresponding to the task starts at the time of its generation t_1 and ends at the time of $x^i_{t_1}$ completion of processing $t_1 + T^{\text{total}}_{i,t_1}$. $x^i_{t_1}$ After a brief delay, the intelligent agent t_1' observes the state at all times $s(x^i_{t_1})$ and performs an offloading action on the task $a(x^i_{t_1})$. When $x^i_{t_1}$ is processed completing, reward feedback for environmental is $r(x^i_{t_1})$. Multiple tasks are processed in parallel.

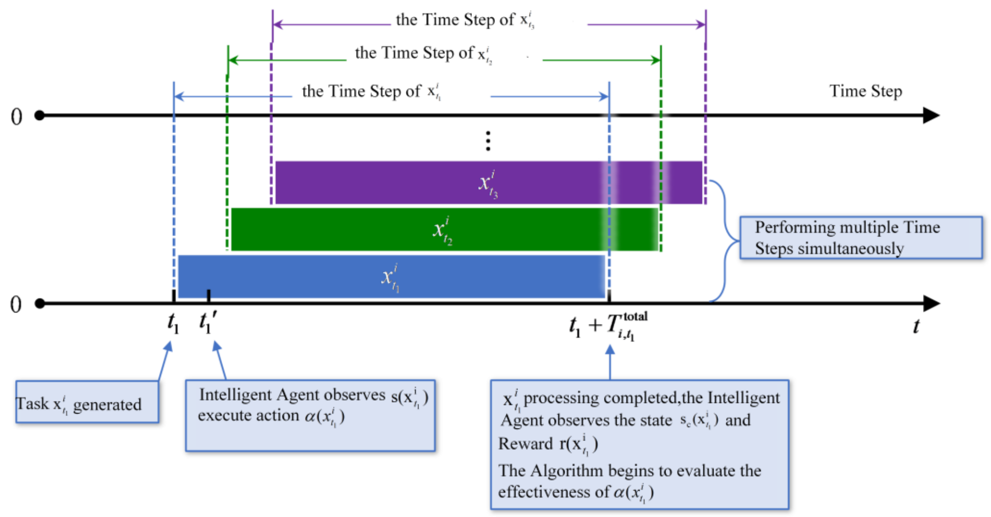

the Time Step of $x^i_{t_3}$
the Time Step of $x^i_{t_2}$
the Time Step of $x^i_{t_1}$
Time Step
0
$x^i_{t_3}$
$x^i_{t_2}$
$x^i_{t_1}$
Performing multiple Time Steps simultaneously
0
t_1
t_1'
$t_1 + T^{\text{total}}_{i,t_1}$
t
Task $x^i_{t_1}$ generated
Intelligent Agent observes $s(x^i_{t_1})$ execute action $\alpha(x^i_{t_1})$
$x^i_{t_1}$ processing completed,the Intelligent Agent observes the state $s_c(x^i_{t_1})$ and Reward $r(x^i_{t_1})$
The Algorithm begins to evaluate the effectiveness of $\alpha(x^i_{t_1})$

Figure 6. Task-oriented Markov decision process.

The TOMDP model includes elements such as state, action, and reward, defined as follows:

(1) Status

The state information is composed of the observations of the node itself and the observations of all nodes it can connect to. Assuming that the node en_j receives the x^i_t summary information of the task at a certain moment, define the relevant x^i_t state as $s(x^i_t)$. We have the following.

$$s(x^i_t) \triangleq \left[\frac{\lambda_{i,t}}{\mu_2}, \frac{\tau_{i,t}}{\mu_3}, \boldsymbol{TP}_{i,t}, \boldsymbol{H}_i, \frac{q^{\text{cache}}_i + q^{\text{comp}}_i + q^{\text{tran}}_i}{R^{\text{ud}}}, \frac{q^{\text{en-comp}}_j + q^{\text{en-tran}}_j}{R^{\text{en}}}, \frac{j}{n}, \boldsymbol{Link}_j, th_{min}, th_{max} \right] \tag{14}$$

where $\lambda_{i,t}$ represents the amount of input data for x^i_t. $\tau_{i,t}$ represents the maximum tolerance for delay for x^i_t. Constants μ_2 and μ_3 are scaling factors used to achieve normalization effects. Vector $\boldsymbol{TP}_{i,t}$ represents the type of task. The matrix $\boldsymbol{H}_i$ represents the en_j user coordinate records stored in. j represents the objective function of strategy learning. The vector $\boldsymbol{Link}_j$ represents the en_j connectivity with other nodes. The capacities of the ud_i cache area, calculation queue, transmission queue, and total cache are q^{cache}_i, q^{comp}_i, q^{tran}_i, and R^{ud}, respectively. The capacities en_j of the calculation queue, transmission queue, and total cache are $q^{\text{en-comp}}_j$, $q^{\text{en-tran}}_j$, and R^{en}, respectively. The minimum and maximum thresholds for the queue are th_{min} and th_{max}. Define the "completion state" observed by en_j when $x^i_{t_1}$ finishes processing as $s_c(x^i_t)$.

(2) Action

The decision vector includes the offloading target selection of the task $\boldsymbol{D}_{i,t}$, computing power allocation weight $ff^j_{i,t}$, queue priority weight $qf_{i,t}$, bandwidth allocation weight $bf_{i,j}$, and thresholds th_{min} and th_{max} in the queue. Assuming the x_t^i offloading goal of the task is en_j, define the x_t^i action related to it as $a\left(x_t^i\right)$. Specifically, it is expressed as follows:

$$a(x_t^i) \triangleq \left[\boldsymbol{D}_{i,t}, ff_{i,t}^j, qf_{i,t}, bf_{i,j}, th_{min}, th_{max}\right] \tag{15}$$

(3) Rewards

The definition of rewards x_t^i in this paper is as follows:

$$R(x_t^i) \triangleq 1 - \frac{Z(x_t^i)}{\frac{\sum_{t' \in T_i^{\text{task}}} Z(x_{t'}^i)}{\sum_{t' \in T_i^{\text{task}}} 1}} + \frac{1}{2} - PEN(x_t^i) \tag{16}$$

where $\frac{\sum_{t' \in T_i^{\text{task}}} Z\left(x_{t'}^i\right)}{\sum_{t' \in T_i^{\text{task}}} 1}$ represents the ud_i average cost of all generated tasks. $PEN\left(x_t^i\right) \in \{0, 1\}$ indicates the penalty value of x_t^i given by the decision constraint module for the initial decision.

4.5. Neural Network Structure Used by TOMAC-PPO

In order to enable intelligent agents to have a certain degree of memory and solve the local observation problem in network models, the TOMAC-PPO algorithm proposed in this paper combines the Transformer model with a fully connected network. The strategy network structure used by TOMAC-PPO is shown in Figure 7. The network input is state s, and the output is action probability density $\pi\left(a\middle|s;\boldsymbol{\theta}^j\right)$, where $\boldsymbol{\theta}^j$ represents the j policy network parameters of the agent. Firstly, the strategy network inputs s into the Transformer model. Next, learn the mapping relationship between state information and action probability distribution through three fully connected layers, and finally output the probability corresponding to each action in that state. The final fully connected (FC) layer specifically adopts the Softmax activation function to ensure that the output value probabilistically satisfies the definition of the probability density function. Similarly, the value network also adopts an almost identical structure, as shown in Figure 8. The goal of the value network is to fit the S state value of the current state $V_\pi(s)$, so only one output value is needed $v\left(s;\boldsymbol{\omega}^j\right)$ (where $\boldsymbol{\omega}^j$ represents the agent's j value network parameters). The structure of the target value network is completely consistent with the value network, and its parameters are represented as $\hat{\boldsymbol{\omega}}^j$.

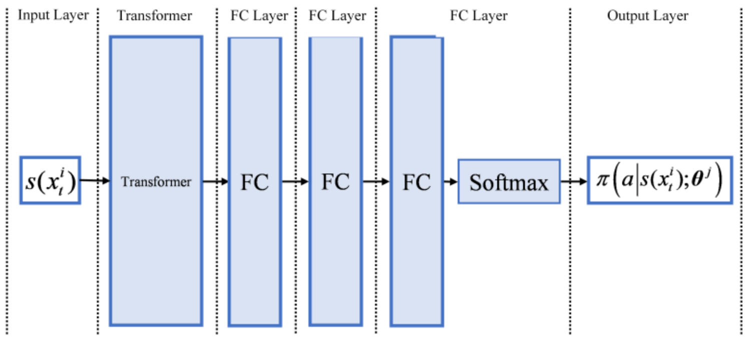


Figure 7. Policy network structure.

4.6. TOMAC-PPO Algorithm Process

The PPO algorithm, proposed by Open AI and DeepMind, is widely regarded as one of the most successful DRL algorithms due to its excellent performance, high efficiency, and stable characteristics [21]. PPO is a strategy learning method, based on the Actor–Critic framework, which can be well adapted to the multi-agent system proposed in this paper. Therefore, this paper combines PPO with Transformer and applies it to the TOMDP model and multi-agent system built in this paper, ultimately forming the TOMAC-PPO algorithm framework. During the training process of the TOMAC-PPO algorithm, constant $\boldsymbol{\theta}_{\text{now}}$ is used to represent the current parameters of the policy network. $\boldsymbol{\theta}$ indicates the parameters of the policy network during the next update, which is the optimization variable of the algorithm. The objective function in TOMAC-PPO theory is denoted as $J(\boldsymbol{\theta})$. Due to the $J(\boldsymbol{\theta})$ difficulty in obtaining the expression, when $\boldsymbol{\theta}$ is in the confidence domain $\mathcal{N}(\boldsymbol{\theta}_{\text{now}})$, an expression $L(\boldsymbol{\theta})$ can be constructed that is close enough to approximate $J(\boldsymbol{\theta})$ and easy to solve, and $L(\boldsymbol{\theta})$ can be used instead of $J(\boldsymbol{\theta})$ as the approximate objective function to solve [22].

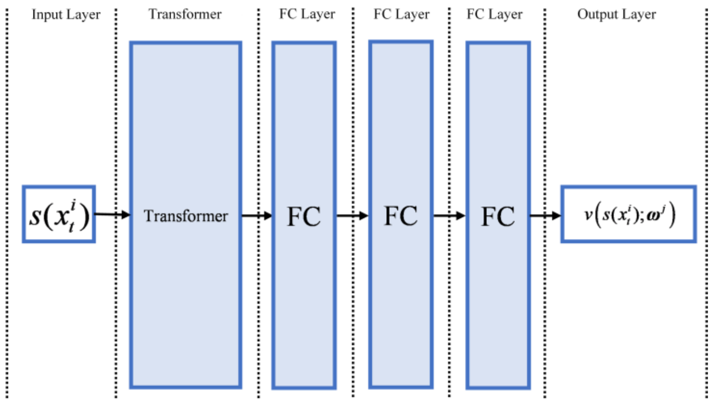


Figure 8. Value network structure.

To satisfy the confidence domain constraint $\boldsymbol{\theta} \in \mathcal{N}(\boldsymbol{\theta}_{\text{now}})$, the divergence needs to be used to measure and limit the difference between the new strategy $\pi(A|S;\boldsymbol{\theta})$ and the old strategy $\pi(A|S;\boldsymbol{\theta}_{\text{now}})$. The TOMAC-PPO algorithm proposes a scheme of approximate optimization objective pruning. The approximate objective function $L(\boldsymbol{\theta})$ is clipped so that $\boldsymbol{\theta}$ does not exceed the confidence domain $\mathcal{N}(\boldsymbol{\theta}_{\text{now}})$ when the gradient ascent algorithm is run. In this scheme, the constraints in the objective function are removed, and the constraint function on $\boldsymbol{\theta}$ is retained, which makes the optimization problem easier to solve.

Here, $s\left(x_t^i\right)$, $a\left(x_t^i\right)$ and $r\left(x_t^i\right)$ will be simplified as s_t, a_t and r_t.

Note that $ratio_t(\boldsymbol{\theta}) = \frac{\pi(a_t|s_t;\boldsymbol{\theta})}{\pi(a_t|s_t;\boldsymbol{\theta}_{\text{now}})}$; the derivation process in reference [23] yields the approximate objective function $L^{\text{CLIP}}(\boldsymbol{\theta})$ after being trimmed by the approximate optimization target clipping method in the TOMAC-PPO algorithm.

$$L^{\text{CLIP}}(\boldsymbol{\theta}) = \mathbb{E}_t\{min[ratio_t(\boldsymbol{\theta}) \cdot Adv_t,\ \text{clip}(ratio_t(\boldsymbol{\theta}), 1-\varsigma, 1+\varsigma) \cdot Adv_t + c \cdot H(s_t;\boldsymbol{\theta})]\} \tag{17}$$

where ς and c are hyperparameters. $\text{clip}(ratio_t(\boldsymbol{\theta}), 1-\varsigma, 1+\varsigma)$ means that the maximum value of $ratio_t(\boldsymbol{\theta})$ is truncated in the interval $\boldsymbol{\theta} \in [1-\varsigma, 1+\varsigma]$, so that the $L^{\text{CLIP}}(\boldsymbol{\theta})$ function must have a global maximum value in the interval $[1-\varsigma, 1+\varsigma]$ as shown in Figure 9. $H(s_t;\boldsymbol{\theta})$ is based on the entropy reward introduced in references [17,22], which can encourage agents to fully explore more actions and avoid premature convergence of strategies to local optima. The advantage function Adv_t can be expressed as follows:

$$Adv_t = Q_\pi(s_t, a_t) - V_\pi(s_t) \tag{18}$$

It is hard to find the specific values of Q_π and V_π in actual training, so it is necessary to approximate Adv_t. Firstly, TOMAC-PPO causes the agent to collect the T step trajectory $\{s_t, a_t, r_t\}$ with the current strategy $\pi(A|s_t;\boldsymbol{\theta}_{\text{now}})$ so that it can get a discount return $u_t = r_t + \gamma \cdot r_{t+1} + \cdots + \gamma^{T-t} \cdot r_T$. Then, in (18), the $Q_\pi(s_t, a_t)$ approximation is replaced by u_t, and the $V_\pi(s_t)$ approximation is replaced by the value network output $v(s_t;\boldsymbol{\omega})$, so that the approximate advantage Adv'_t can be obtained:

$$Adv'_t = r_t + \gamma \cdot r_{t+1} + \cdots + \gamma^{T-t} \cdot r_T - v(s_t;\boldsymbol{\omega}) \tag{19}$$

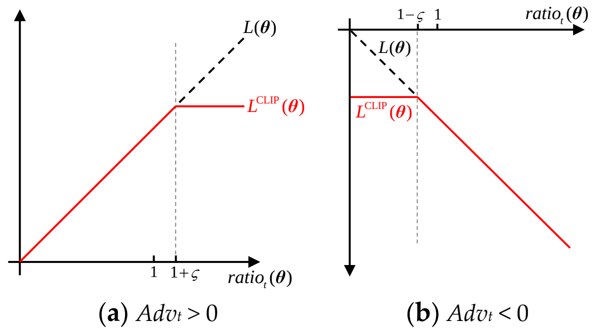


Figure 9. Clipped summarize objective schematic diagram.

So far, the variables in the expression of the approximate objective function $L^{\text{CLIP}}(\boldsymbol{\theta})$ can be obtained, and the gradient ascent method can be used to continuously update the strategic network parameter $\boldsymbol{\theta}$ of each agent to increase $L^{\text{CLIP}}(\boldsymbol{\theta})$.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{now}} + \beta \cdot \nabla_{\boldsymbol{\theta}} L^{\text{CLIP}}(\boldsymbol{\theta}_{\text{now}}) \tag{20}$$

where β is the learning rate, which is the training value network, and the value network loss function is defined as follows:

$$L^{V}(\boldsymbol{\omega}) \triangleq -\sum_{t=1}^{T} (Adv'_t)^2 \tag{21}$$

The larger the approximate advantage Adv'_t in (21), the smaller the loss of the value network. The purpose of defining the loss function in this way is to encourage the agent to try to increase the dominance value and choose the action that increases the dominance as much as possible. The value network parameter $\boldsymbol{\omega}$ can be updated by the gradient descent method to reduce the loss $L^{V}(\boldsymbol{\omega})$:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}_{\text{now}} - \alpha \cdot \nabla_{\boldsymbol{\omega}} L^{V}(\boldsymbol{\omega}_{\text{now}}) \tag{22}$$

where α is the learning rate. The target value network parameter $\hat{\boldsymbol{\omega}}$ can be updated in a similar way as shown in (23).

$$\hat{\boldsymbol{\omega}} \leftarrow \sigma \cdot \boldsymbol{\omega} + (1-\sigma) \cdot \hat{\boldsymbol{\omega}} \tag{23}$$

where parameter $\sigma \in (0,1)$ is the update ratio for the target network.

The specific steps of the TOMAC-PPO algorithm are shown in Algorithm 1.

Algorithm 1. Task-Oriented Multi-Agent Collaborative-Proximal Policy Optimization (TOMAC-PPO)

```
Input: Training rounds $e^{max}$, pruning parameters $\varsigma$, value network $\alpha$ learning rate, strategy network learning rate $\alpha$, discount rate $\gamma$, target value network update ratio $\sigma$.
Output: Optimal task offloading and resource allocation strategy $\pi(a|s;\boldsymbol{\theta}^j)$.
1. Randomly initialize the parameter $\boldsymbol{\theta}$ of each strategy network, as well as each value network parameter $\boldsymbol{\omega}$, and the target value network parameter $\hat{\boldsymbol{\omega}}$;
2. For $episode = 1, 2, \ldots, e^{max}$ do
3.   For all agents $j$, where $\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n}$ do in parallel
4.     According to the current strategy $\pi(A|s_t;\boldsymbol{\theta}^j_{\text{now}})$, collect $T$ step trajectory $\{s_t, a_t, r_t\}$;
5.     According to (19), calculate the approximate advantage $Adv'_t$;
6.     Update the policy network parameter $\boldsymbol{\theta}^j$ of agent $j$ according to (20);
7.     According to (22) and (23), the value network parameter $\boldsymbol{\omega}^j$ and the target value network parameter $\hat{\boldsymbol{\omega}}^j$ of $j$ are updated.
   End for
End for
```

In the following, we give the complexity analysis of the TOMAC-PPO algorithm. The main time spent on this algorithm is in the second step, which is a training round loop with e^{max} training rounds and a time complexity of O (e^{max}). The third step is that the agent operates in parallel, with a time complexity of O (n). So, the overall time complexity is O (ne^{max}).

5. Experimental Results and Analysis

To evaluate the optimization effect of the proposed scheme on MEC network performance, we compare the following four offloading decision schemes with TOMAC-PPO.

(1) TOMAC-A2C (Task-Oriented Multi-Agent Cooperative Advantage Actor–Critic). A2C algorithm is one of the classic strategy learning algorithms in the RL field.
(2) TO-A3C (Task-Oriented Asynchronous Advantage Actor–Critic). TO-A3C belongs to the parallel RL method and does not use multi-agent systems. The A3C algorithm improves its performance by establishing multiple independent single agent A2C training environments, enabling them to train in parallel [24].
(3) CCP (Cloud Computing Priority). CCP adopts the principle of "deliver tasks to upper level processing as much as possible", and prioritizes offloading all tasks to the cloud for processing.
(4) LC (Local Computing). After the task is generated, skip the information reporting process and directly calculate locally by the user.

We build the algorithm environment for active queue management, offloading scheduling rules, and multi-agent systems described in this paper to run the TOMAC-PPO algorithm. We give convergence analysis, and test average cost, average delay, average energy consumption, and drop rate.

5.1. Experimental Environment and Parameter Settings

The experimental environment is implemented using Python language, and the real dataset used includes the longitude and latitude of mobile users and the maximum CPU frequency as user device data. This paper stipulates that the probabilities of users generating high-priority tasks, critical tasks, low-priority tasks, and regular tasks are [0.2, 0.2, 0.2, 0.4], respectively. The specific parameters are shown in Table 1.

5.2. Convergence Analysis

For 50 users and a probability of failure $p_{en} = p_{fiber}$ = 0.1%, we give convergence results for the TOMAC-A2C algorithm and the TOMAC-PPO algorithm.

The cumulative reward convergence of TOMAC-A2C and TOMAC-PPO during training is shown in Figure 10. We can see that the moving average reward curves of both algorithms gradually converge with the increase in training episodes, proving the effectiveness of TOMAC-A2C and TOMAC-PPO algorithms. TOMAC-PPO not only has a significant advantage in convergence speed compared to TOMAC-A2C but also improves the final convergence value by about 23.9% compared to MAC-A2C, demonstrating its outstanding performance advantage. From the stability of the convergence curve, although the final convergence value of TOMAC-PPO is significantly higher than TOMAC-A2C, its vibration amplitude is also obviously larger. This may be a phenomenon caused by the introduction of entropy in the approximate objective function $L^{\text{CLIP}}(\boldsymbol{\theta})$ of TOMAC-PPO, which leads to a higher desire for agents to explore randomly.

5.3. Optimization Performance Evaluation

To evaluate the performance of various offloading schemes under different network load levels, this experiment tested the variation of average task cost with the number of users under the $p_{en} = p_{fiber}$ = 0.1% setting of failure probability. As shown in Figure 11, the TOMAC-PPO method always maintains the lowest task overhead under various load conditions, and its performance advantage continues to expand with the increase in the number of users. When the number of users is 50, TOMAC-PPO reduces the average cost by 19.4% to 66.6% compared to other solutions.

Table 1. Parameters setting.

Symbol	Meaning	Value
n	Number of edge nodes	10
R^{ud}	User device cache capacity	49, 152 Mbit
R^{en}	Edge node cache capacity	262, 144 Mbit
B	Base station bandwidth resources	20 MHz
r	Base station service radius	100 m
p^{edge}	Base station transmission power	200 W
p^{user}	User device transmission power	0.2 W
κ	User device energy efficiency coefficient	$U\,(4.13, 66.16) \times 10^{-27}$
σ^2	Gaussian noise power	1.5×10^{-8} W
V_{space}	The propagation rate of electromagnetic waves in the air	3×10^8 m/s
V_{fiber}	The propagation rate of electromagnetic waves in a circuit	2×10^8 m/s
C_{fiber}	The transmission rate of wired communication	1000 Mbit/s
T_{repair}	Fault repair duration	$U(10, 60)$ s
λ	Task input data volume	$N(5100)$ Mbit
λ^{out}	Task calculation result data volume	$N(1, 50)$ Mbit
ρ	Task computing density	$N(0.297, 0.1)$ G.c./Mbit
num^{core}	Number of CPU cores on edge servers	14
f^{edge}	Edge server CPU frequency	2.4 GHz
f^{cloud}	Cloud server CPU frequency	10 GHz
T^{cloud}_{delay}	Core network forwarding delay	$N(50, 15)$ ms
d_{edge}	The distance between adjacent nodes	150 m
p_{drop}	RED Dropped Task Probability	1/50
α, β	Learning rate	3×10^{-4}
e^{max}	Training epochs	1000
γ	Discount rate	0.95
σ	Target network update ratio	0.01
ς	Crop parameters	0.2
c	Entropy weight	0.01

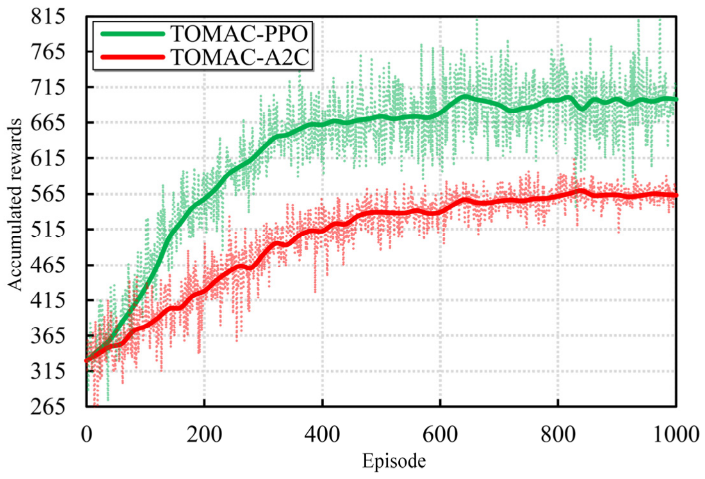


Figure 10. Offloading and allocation algorithms conversion and cumulative rewards versus episodes.

To test the response capability of various offloading schemes to line faults, this experiment tested the average task cost with the variation of the failure probability of edge servers and lines under the setting of a large number of users (50).

As shown in Figure 12, when the failure rate is less than 70%, the task overhead corresponding to TOMAC-PPO and TOMAC-A2C methods is significantly lower than other schemes. When the failure rate reaches over 70% and continues to increase, the task overhead of TOMAC-PPO and TOMAC-A2C increases sharply and gradually approaches the curve of the LC scheme, and the gap between the two also decreases. When the failure rate reaches 100%, it is equivalent to the entire edge layer device being in a disconnected state. At this time, except for the LC scheme, the average cost of the other schemes is around 2.3, slightly higher than the average cost of the LC scheme, which is 2.24. This indicates that, when the failure rate is 100%, each optimization plan has no other countermeasures besides performing local calculations. Due to the fact that users still need to attempt to report task information at this time, the cost incurred is slightly higher than that of the LC scheme. When the failure rate is less than 50%, the cost curve of the CCP scheme shows a downward trend. This is because line failures force more tasks to be handed over to edge layers with lower latency, reducing unnecessary tasks from cloud migration. When the failure rate exceeds 50%, the CCP overhead curve gradually increases and approaches the LC scheme, because the reduction in the number of available servers results in a large number of tasks that cannot be offloaded and can only be processed at the user layer. In the vast majority of failure rate scenarios, the TOMAC-PPO scheme can maintain the lowest average cost, which proves that compared to other optimization schemes, TOMAC-PPO has better robustness in dealing with various failure situations.

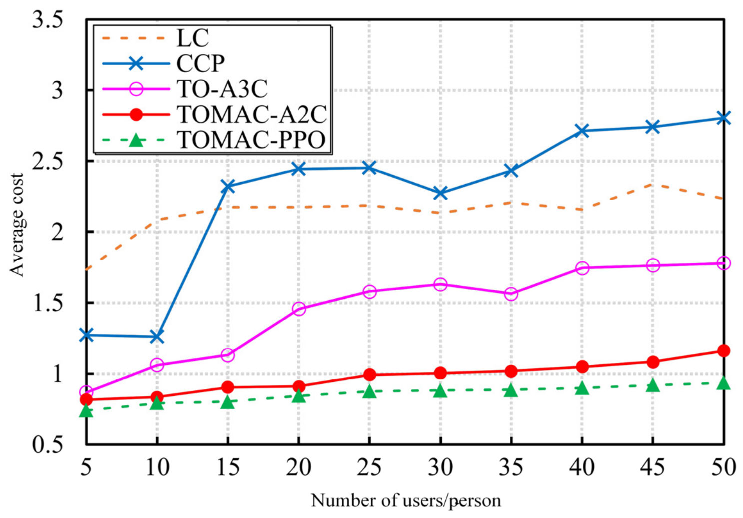


Figure 11. Average task cost versus user number.

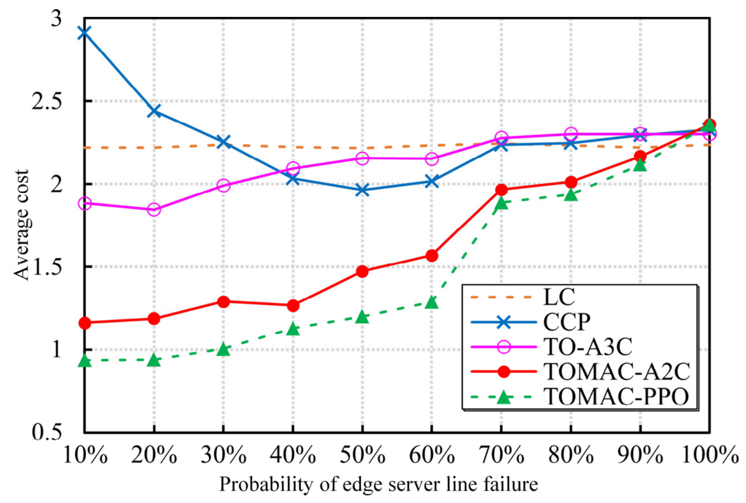
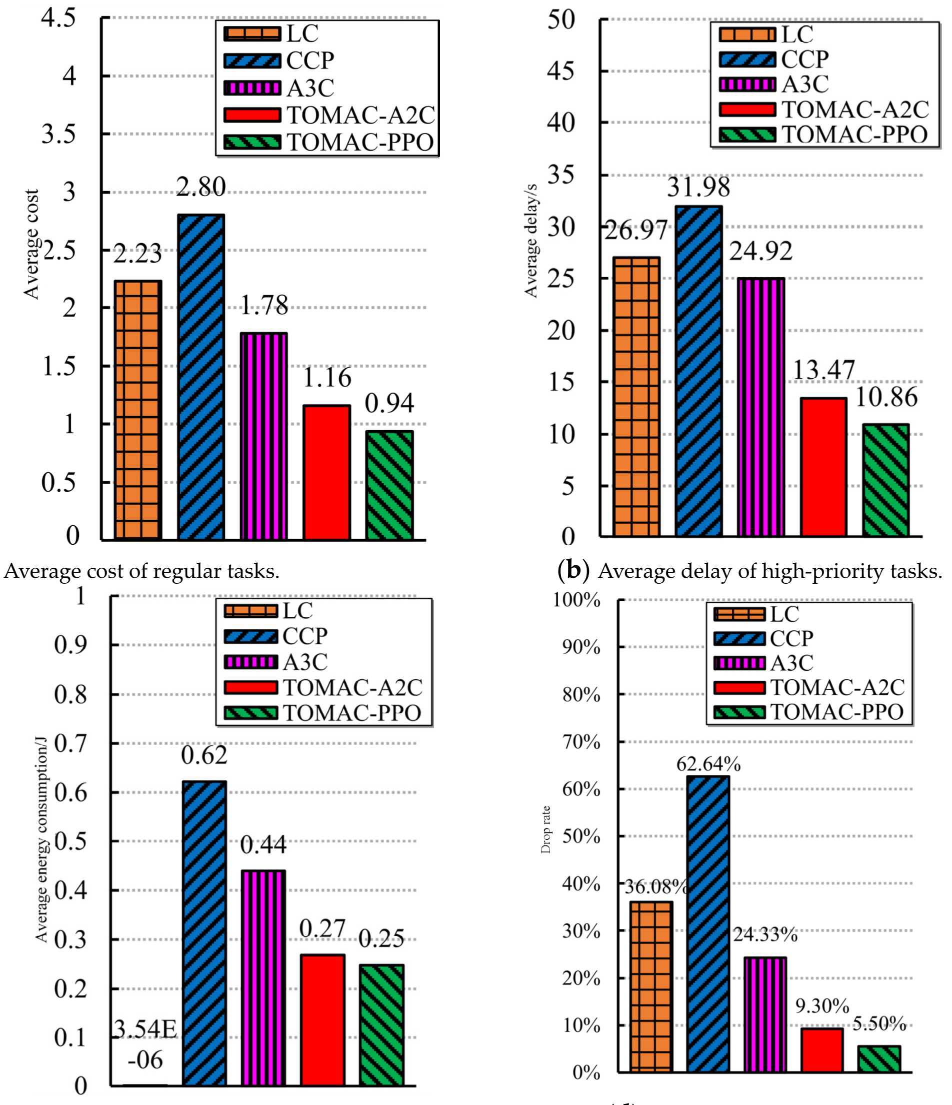


Figure 12. Average task cost versus failure probability.

To test whether each offloading scheme adopts targeted offloading processing for different types of tasks, we test the processing effect of each scheme on different types of tasks under the conditions of 50 users and failure probability with $p_{en} = p_{fiber} = 0.1\%$, as shown in Figure 13.



(**a**) Average cost of regular tasks. (**b**) Average delay of high-priority tasks.

(**c**) Average energy consumption of low-priority tasks. (**d**) Critical task drop rate.

Figure 13. Optimization effect of schemes on the key performance metrics of different types of tasks.

As shown in the figure, besides energy consumption, the TOMAC-PPO method can achieve the best optimization results in various evaluation indicators that are important for tasks. Due to the significant energy-saving advantages of the LC scheme, the TOMAC-PPO method falls short in optimizing energy consumption for low-priority tasks compared to LC. However, TOMAC-A2C not only focuses on the energy consumption of this type of task but also considers the optimization of the task dropout rate. If too many low-priority tasks are delegated to local processing, it may result in a high dropout rate due to weak local computing power. Therefore, TOMAC-A2C generates more energy consumption by balancing these two evaluation indicators. Except for LC, TOMAC-PPO still has the best energy optimization effect on low-priority tasks. The results demonstrate that the TOMAC-PPO scheme can more accurately respond to diverse task requirements compared to other schemes when dealing with different types of tasks.

5.4. Experiment Results Discussion

In the above experiments, we compare the four offloading decision schemes with TOMAC-PPO at average cost, average delay, average energy consumption, and drop rate.

TOMAC-PPO proposed can effectively reduce the average task cost in MEC network environments with different numbers of users and system failure probabilities. Compared to the other four baseline methods, the proposed scheme has better optimization performance in the vast majority of cases, and its advantages are particularly prominent in dealing with high-load situations. For example, the proposed TOMAC-PPO can reduce the average cost by from 19.4% to 66.6% compared to other offloading schemes under the same network load. In addition, the drop rate of some baseline algorithm with 50 users can achieve 62.5% for critical task, while the proposed TOMAC-PPO only has 5.5%. The main reasons are that (1) multi-agent parallel offloading and task allocation are introduced in the TOMAC-PPO and (2) active queue management and offloading scheduling rules have been implemented for different tasks. Certainly, the proposed TOMAC-PPO has some limitations: (1) With the continuous increase in network scale, the scalability of multi-agent systems will be challenged. The information synchronization protocol used within the system can further reduce communication overhead, and the information lag of large-scale systems is also a major issue that needs to be addressed. (2) Due to various limitations, this paper can only use Python code to implement a simulation system for simple simulation of MEC networks. In future research, professional simulation software can be further used to achieve more realistic simulations, and consideration can be given to MEC computation offloading in more scenarios such as drones and connected vehicles.

6. Conclusions and Further Works

Task offloading and resource allocation is a research hotspot in cloud-edge collaborative computing. This paper constructs a cloud-edge collaborative computing model, and related task queue, delay, and energy consumption model, and gives joint optimization problem modeling for task offloading and resource allocation with multiple constraints. Furthermore, it designs a decentralized task offloading and resource allocation scheme based on "task-oriented" multi-agent reinforcement learning. In this scheme, we present information synchronization protocol and offloading scheduling rules and use edge servers as agents to construct a multi-agent system based on the Actor–Critic framework. The proposed TOMAC-PPO applies the proximal policy optimization to the multi-agent system and combines the Transformer neural network model to realize the memory and prediction of network state information. Experimental results show that this algorithm has better convergence speed and can effectively reduce the service cost, energy consumption, and task drop rate under high load and high failure rates. For example, the proposed TOMAC-PPO can reduce the average cost by from 19.4% to 66.6% compared to other offloading schemes under the same network load. In addition, the drop rate of some baseline algorithms with 50 users can achieve 62.5% for critical tasks, while the proposed TOMAC-PPO only has 5.5%.

In future works, due to the innovation of the TOMDP dynamic time slot model proposed in this paper, it conflicts with existing DRL code patterns, resulting in certain encoding difficulties. Therefore, how to be compatible with existing DRL code frameworks is an area that TOMDP needs to optimize. In addition, the design of states, actions, and rewards in TOMDP is also a decisive factor in the optimization effect of the algorithm and can be further improved.

Author Contributions: Conceptualization, G.J.; methodology and validation, R.H.; investigation and writing—original draft preparation, G.W. and Z.B.; writing—review and editing, R.H. and G.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China. No. 62062007, 2022 Guangxi Vocational Education Teaching Reform Research Key Project, and Research on the Six in One Talent Training Model of "Post, Course, Competition, Certificate, Training, and Creation" in the College of Artificial Intelligence Industry, No. GXGZJG2022A011. Design and Application of Computer Experiment Site Management System Based on Information Technology, 2023YKYZ001.

Data Availability Statement: Data are available upon request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

TOMAC-PPO	Task-Oriented Multi-Agent Collaborative-Proximal Policy Optimization
PPO	Proximal Policy Optimization
IoT	Internet of Thing
MEC	Mobile Edge Computing
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
UAVs	Unmanned Aerial Vehicles
DQN	Deep Q Network
MARL	Multi-Agent Reinforcement Learning
MADDPG	Multi-Agent Deep Determining Policy Gradient
FIFO	First In First Out
FC	Fully Connected
TOMAC-A2C	Task-Oriented Multi-Agent Cooperative Advantage Actor–Critic
CCP	Cloud Computing Priority
LC	Local Computing

References

1. IoT and Non-IoT Connections Worldwide 2010–2025. Available online: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide (accessed on 28 August 2024).
2. IoT Is Not a Buzzword but Necessity. Available online: https://www.3i-infotech.com/iot-is-not-just-a-buzzword-but-has-practical-applications-even-in-industries/ (accessed on 28 August 2024).
3. Zhang, Y.-L.; Liang, Y.-Z.; Yin, M.-J.; Quan, H.-Y.; Wang, T.; Jia, W.-J. Survey on the Methods of Computation Offloading in Molile Edge Computing. *J. Comput. Sci. Technol.* **2021**, *44*, 2406–2430.
4. Duan, S.; Wang, D.; Ren, J.; Lyu, F.; Zhang, Y.; Wu, H.; Shen, X. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Commun. Surv. Tutor.* **2022**, *25*, 591–624. [CrossRef]
5. Hua, H.; Li, Y.; Wang, T.; Dong, N.; Li, W.; Cao, J. Edge computing with artificial intelligence: A machine learning perspective. *ACM Comput. Surv.* **2023**, *55*, 1–35. [CrossRef]
6. Kar, B.; Yahya, W.; Lin, Y.-D.; Ali, A. Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 1199–1226. [CrossRef]
7. Arjona-Medina, J.A.; Gillhofer, M.; Widrich, M.; Unterthiner, T.; Brandstetter, J.; Hochreiter, S. RUDDER: Return decomposition for delayed rewards. In Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; pp. 13544–13555.
8. Zhang, X.-C.; Ren, T.-S.; Zhao, Y.; Rui, F. Joint Optimization Method of Energy Consumption and Time Delay for Mobile Edge Computing. *J. Univ. Electron. Sci. Technol. China* **2022**, *51*, 737–742.
9. Wu, H.-Y.; Chen, Z.-W.; Shi, B.-W.; Deng, S.; Chen, S.; Xue, X.; Feng, Z. Decentralized Service Request Dispatching for Edge Computing Systems. *Chin. J. Comput.* **2023**, *46*, 987–1002.
10. Ma, L.; Wang, X.; Wang, X.; Wang, L.; Shi, Y.; Huang, M. TCDA: Truthful combinatorial double auctions for mobile edge computing in industrial internet of things. *IEEE Trans. Mob. Comput.* **2021**, *21*, 4125–4138. [CrossRef]
11. Cang, Y.; Chen, M.; Pan, Y.; Yang, Z.; Hu, Y.; Sun, H.; Chen, M. Joint user scheduling and computing resource allocation optimization in asynchronous mobile edge computing networks. *IEEE Trans. Commun.* **2024**, *72*, 3378–3392. [CrossRef]
12. Peng, Z.; Wang, G.; Nong, W.; Qiu, Y.; Huang, S. Task offloading in multiple-services mobile edge computing: A deep reinforcement learning algorithm. *Comput. Commun.* **2023**, *202*, 1–12. [CrossRef]
13. Li, J.; Yang, Z.; Wang, X.; Xia, Y.; Ni, S. Task offloading mechanism based on federated reinforcement learning in mobile edge computing. *Digit. Commun. Netw.* **2023**, *9*, 492–504. [CrossRef]
14. Li, Y.; Aghvami, A.H.; Dong, D. Path Planning for Cellular-Connected UAV: A DRL Solution with Quantum-Inspired Experience Replay. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 7897–7912. [CrossRef]
15. Li, Y.; Aghvami, A.H. Radio Resource Management for Cellular-Connected UAV: A Learning Approach. *IEEE Trans. Commun.* **2023**, *71*, 2784–2800. [CrossRef]
16. Kuang, Z.-F.; Chen, Q.-L.; Li, L.-F.; Deng, X.H.; Chen, Z.G. Multi-user edge computing task offloading scheduling and resource allocation based on deep reinforcement learning. *Chin. J. Comput.* **2022**, *45*, 812–824. (In Chinese)
17. Tuli, S.; Ilager, S.; Ramamohanarao, K.; Buyya, R. Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks. *IEEE Trans. Mob. Comput.* **2020**, *21*, 940–954. [CrossRef]

18. Zhang, K.; Yang, Z.; Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. In *Handbook of Reinforcement Learning and Control*; Springer: New York, NY, USA, 2021; pp. 321–384.
19. Zhang, P.; Tian, H.; Zhao, P.; He, S.; Tong, Y. Computation offloading strategy in multi-agent cooperation scenario based on reinforcement learning with value-decomposition. *J. Commun.* **2021**, *42*, 1–15. (In Chinese)
20. Cao, Z.; Zhou, P.; Li, R.; Huang, S.; Wu, D. Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0. *IEEE Internet Things J.* **2020**, *7*, 6201–6213. [CrossRef]
21. Wang, Y.; He, H.; Tan, X. Truly proximal policy optimization. In Proceedings of the 35th Uncertainty in Artificial Intelligence Conference (UAI), Tel Aviv, Israel, 23–25 July 2019; PMLR: New York, NY, USA, 2020; Volume 115, pp. 113–122.
22. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the 32nd International Conference on Machine Learning (ICML), Lille, France, 6–11 July 2015; PMLR: New York, NY, USA, 2015; Volume 37, pp. 1889–1897.
23. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347. [CrossRef]
24. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.P.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML), New York City, NY, USA, 19–24 June 2016; PMLR: New York, NY, USA, 2016; Volume 48, pp. 1928–1937.