future internet

MDPI

Article

MADDPG-Based Offloading Strategy for Timing-Dependent Tasks in Edge Computing

Yuchen Wang [1,2], Zishan Huang [1,2], Zhongcheng Wei [1,2] and Jijun Zhao [1,2,*]

[1] School of Information and Electrical Engineering, Hebei University of Engineering, Handan 056038, China; ssdlhy123450@163.com (Y.W.); huangzseulgi@163.com (Z.H.); weizhongcheng@hebeu.edu.cn (Z.W.)
[2] Hebei Key Laboratory of Security and Protection Information Sensing and Processing, Handan 056038, China
* Correspondence: zjijun@hebeu.edu.cn

Abstract: With the increasing popularity of the Internet of Things (IoT), the proliferation of computation-intensive and timing-dependent applications has brought serious load pressure on terrestrial networks. In order to solve the problem of computing resource conflict and long response delay caused by concurrent application service applications from multiple users, this paper proposes an improved edge computing timing-dependent, task-offloading scheme based on Multi-Agent Deep Deterministic Policy Gradient (MADDPG) that aims to shorten the offloading delay and improve the resource utilization rate by means of resource prediction and collaboration among multiple agents to shorten the offloading delay and improve the resource utilization. First, to coordinate the global computing resource, the gated recurrent unit is utilized, which predicts the next computing resource requirements of the timing-dependent tasks according to historical information. Second, the predicted information, the historical offloading decisions and the current state are used as inputs, and the training process of the reinforcement learning algorithm is improved to propose a task-offloading algorithm based on MADDPG. The simulation results show that the algorithm reduces the response latency by 6.7% and improves the resource utilization by 30.6% compared with the suboptimal benchmark algorithm, and it reduces nearly 500 training rounds during the learning process, which effectively improves the timeliness of the offloading strategy.

Keywords: edge computing; task dependency; computational offloading; resource prediction; deep reinforcement learning; multi-agent collaboration

Citation: Wang, Y.; Huang, Z.; Wei, Z.; Zhao, J. MADDPG-Based Offloading Strategy for Timing-Dependent Tasks in Edge Computing. *Future Internet* **2024**, *16*, 181. https://doi.org/10.3390/fi16060181

Academic Editor: Paolo Bellavista

Received: 16 April 2024
Revised: 3 May 2024
Accepted: 17 May 2024
Published: 21 May 2024

1. Introduction

With the popularization of the Internet of Things (IoT), a large number of computation-intensive applications are emerging, such as face recognition [1] and autonomous driving [2], which increase the pressure on terrestrial networks to transmit and process data. In addition, realistic scenarios such computation-intensive applications often have timing dependencies, which make the network environment more complex [3]. In order to effectively deal with such computation-intensive tasks with timing dependencies, mobile edge computing (MEC) has emerged. Compared with traditional cloud computing [4], MEC has the advantages of low latency, high bandwidth, and agile deployment due to being closer to users. MEC has become an effective tool to improve the processing efficiency of timing-dependent tasks in the network [5].

However, in multi-user concurrent scenarios, MEC technology still faces the constraints of edge server resource limitations and the complexity of inter-task timing dependencies while improving the user response rate. With the generation of a large number of computation-intensive tasks in time and space, the random scheduling of tasks will lead to the localized clustering of tasks on the edge servers, further inducing load imbalance of MEC network nodes [6]. This phenomenon of chaotic task processing order or server paralysis due to excessive access makes it more difficult to ensure the delay in

task offloading and the timeliness of the offloading policy [7,8]. Therefore, in realistic IoT scenarios in which multiple users concurrently apply for services, it is of great significance to formulate a node cooperative offloading strategy that can reduce the task processing delay as well as improve the resource utilization rate to enhance the timeliness of the task offloading strategy.

According to the timing dependencies between subtasks and the dynamic scheduling of node resources, realizing serial and parallel computation between subtasks is the key to developing node-cooperative offloading strategies. Due to the differences in timing dependencies between the subtasks in each application and the unpredictable and dynamic change of resources on the node due to simultaneous offloading of tasks by multiple users, the construction of node-collaborative offloading strategies faces many challenges. Existing research has used deep-reinforcement learning methods to solve the task offloading problem in complex scenarios with dynamically changing computing resources [9]. However, single-agent offloading algorithms suffer from the problems of restricted observation areas and overfitting the training environment, which leads to the algorithms not being able to comprehensively sense the changes in the whole system [10,11]. When the system environment is changing constantly, such algorithms have difficulty guaranteeing the offloading performance.

Based on the above discussion, this paper proposes the MADDPG-based timing-dependent task offloading (MADDPG-DTO) algorithm to reduce the offloading latency and improve the resource utilization rate and the offloading strategy timeliness. The main contributions of this paper are as follows:

- Aiming at the problem of long offloading delay due to concurrent application of services by multiple users and timing dependency constraints among tasks, this paper analyzes timing dependencies based on the depth-first traversal (DFS) algorithm and integrates gated recursive units (GRUs) into a multi-agent reinforcement learning algorithm to memorize effective information about historical offloading decisions and task attributes and predicts computational resources required for subsequent tasks. This method guarantees the supply of computational resources for user demand and reduces the number of iterations of the multi-agent algorithm learning, thus reducing the offloading delay and improving the timeliness of the offloading strategy.
- Aiming at the problem of low resource utilization due to resource conflict caused by the concurrent offloading of tasks by multiple users, an edge-computing, time-dependent task offloading algorithm based on user behaviour prediction is proposed. In a complex environment with large differences in resource conflicts and task time-dependent constraints, the training process of the MADDPG algorithm is improved to avoid a situation in which the local decision-making of single-agent algorithms is difficult to comprehensively optimize the global benefits, reduce the offloading delay, and improve the global resource utilization.
- Simulation results show that the proposed algorithm performs better than the DQN and DDQN single-agent algorithms in terms of both delay reduction and resource utilization improvement. In addition, the algorithm reduces the response latency by 6.7% and improves the resource utilization by 30.6% compared to the suboptimal offloading algorithm based only on MADDPG and reduces nearly 500 training rounds during the learning process, which effectively improves the timeliness of the offloading strategy.

The structure of this paper is organized as follows. Section 2 summarizes relevant work on task offloading. Section 3 describes the construction of a model related to timing-dependent task offloading and models the offloading problem. In Section 4, the proposed user-prediction-based timing-dependent task offloading algorithm is presented. Section 5 analyzes the simulation's experimental results. Finally, the contribution of this paper is summarized in Section 6.

2. Related Works

With the increase of application service requests in IoT, the quality of experience (QoE) provided by the network has become the focus of current research attention, and the requirements for it have become more and more stringent [12]. However, in multi-user concurrent offloading scenarios, the challenges of computational offloading are exacerbated by the diversity of user behaviors, the temporal dependency properties among tasks, and the dynamic instability of node resources. To cope with these challenges, this paper focuses on the dependency task-offloading problem for multi-user concurrency scenarios.

2.1. Timing-Dependent Task Offloading in MEC

Mobile applications generally contain multiple delay-sensitive and computation-intensive timing-dependent tasks which need to follow a strict logical order when being executed. However, many studies mainly design offloading schemes according to the computing resource required for tasks, ignoring the timing dependencies among the subtasks. This offloading approach, which does not consider the execution order, may affect the QoE of users and aggravate the response delay. Especially in multi-user concurrent scenarios, tasks processed by different user requests may affect each other, further aggravating the complexity of task scheduling and resource allocation.

The authors in [12] jointly optimized task dependencies through deadline constraints for delay-sensitive tasks and proposed an edge-computing task optimization strategy based on a directed cyclic graph model and on priority-aware scheduling to achieve the efficient processing of tasks, but they did not consider the impact of resource conflicts and resource constraints on task offloading in multi-user concurrency scenarios. The authors in [13] used the DRL algorithm to train the system to optimize energy efficiency and reduce the time to complete the task, but their use of a single-agent algorithm that only considered its own interests is not applicable to multi-user concurrent scenarios and was prone to training overfitting conditions. The authors in [14] proposed a collaborative scheme for sharing workloads among UAVs to minimize average user latency by jointly controlling task-dependent offloading decisions and allocating UAV communication resources. The authors in [15] proposed a dependency-aware task offloading framework, COFE, where users could offload computationally intensive tasks with dependency constraints to the MEC-cloud system to minimize the average latency of dependent tasks. None of the works in [14,15] considered the problem of the uneven allocation of computational resources affecting offloading efficiency in multi-user scenarios. The authors in [16] proposed COFE, a dependency-aware task offloading framework that allowed users to offload computation-intensive tasks with dependency constraints to the MEC-cloud system. The authors in [17] proposed a deadline-violation minimization computational-offloading scheme based on task migration and merging, which effectively reduced the system's deadline violation rate in an MEC system by means of a multi-priority task ordering algorithm and a deep deterministic policy gradient (DDPG) learning algorithm, but the work did not consider resource allocation among tasks, and the single-agent algorithm used was not applicable to multi-user concurrent offloading environments.

Although all of the above works optimize the task offloading method and improve the task scheduling efficiency to a certain extent, they mainly focus on considering the offloading decision of a single user per unit of time while relatively neglecting the demand of different users on computing nodes in the multi-user concurrent scenarios, and most of them are not applicable to complex offloading scenarios in which the resources of the multi-user concurrently change dynamically. The authors in [18] considered the joint optimization of topology reconstruction and sub-task scheduling to minimize the average completion time of subtasks, but the dynamic allocation of resources was not sufficiently taken into account in this work, which is a little bit less relevant to the actual scenarios. In order to ensure the global computational resource utilization in multi-user concurrent scenarios, the authors in [19] modeled the task offloading decision as a Markov decision process and applied a graph attention network to extract the dependency information of different

tasks while combining Long Short-Term Memory (LSTM) and a Deep Q-learning Network (DQN) to deal with sequential problems. The authors in [20] proposed an enhanced multi-objective version of the orangutan algorithm for offloading interdependent tasks in edge cloud environments, in which multi-objective optimization reduced offloading latency and improves resource utilization. The authors in [21] jointly optimized dependent task offloading, computational resource allocation, user-transmission power control, and channel resource allocation to maximize the quality of user experience by building a weighted aggregate model for reducing latency and energy saving and optimizing a multi-cellular, multi-user task offloading and resource allocation model based on the multi-agent proximity policy. The authors in [22] addressed the triangular dependencies of multiple subtasks and the routing of interdependent subtasks to maximize the total offline revenue. The authors in [23] investigated a lightweight and efficient edge computing task migration scheme for low-power IoT systems. By taking into account the topology/schedule of IoT tasks, heterogeneous resources on the edge servers, and wireless interference, the authors proposed a distributed consensus algorithm to support multi-user migration, which effectively reduced the task execution time and improves resource utilization. In addition, to ensure the quality of experience, achieve low latency, and provide highly reliable services, the authors in [24] proposed a hybrid hierarchical task offloading strategy. On the other hand, the authors in [25] introduced an optimization scheme by implementing a map-based algorithm. The authors in [26] established a system-cost evaluation metric to minimize this metric as an objective for solving the multi-service task MEC problem. However, the above research work on task offloading considering computational resources in a multi-user multi-server environment has not considered the problem of task-timing dependency constraints that exist for user requests in real-world environments, making the formulation of offloading strategies more difficult.

Therefore, subsequent research works need to comprehensively consider the timing dependency constraints among tasks and the task scheduling and resource allocation issues in multi-user concurrent scenarios to better fit actual scenarios and realize more efficient task offloading and scheduling strategies.

2.2. Edge-Computing Task Offloading Based on Resource Prediction

With the large number of local devices accessed in IoT, how to improve the speed of making offloading decisions while guaranteeing the efficiency of user response has become the focus of current researchers. The resource-prediction-based offloading of edge computing tasks can effectively improve the problem of large differences in task execution results due to the diversity of temporal dependencies within the task, while the effective prediction of the computational resources required for subsequent tasks can effectively improve the efficiency of users in formulating offloading strategies. This offloading method based on resource prediction can effectively alleviate the global resource tension and realize the rational allocation of resources.

The authors in [27] targeted a multi-objective task offloading scheme for edge devices that aims to optimize task execution time, device energy consumption, and rental cost, while taking into account the real-time environment of the device and the user's preferences to improve the task completion rate and reduce the associated metrics. The authors in [28] proposed a lightweight mobile prediction and offloading framework using artificial neural networks to reduce the complexity of predicting user location for computationally intensive task offloading. The authors in [29] designed a distributed service offloading D-SOAC method based on deep learning and deep reinforcement learning, which predicted the potential customer demand in the central cloud through deep spatio-temporal residual networks and collaborated with edge servers to solve the problem of how to optimize the service offloading location to reduce the user's service latency under the condition of limited computational resources, which effectively reduces the average service latency of the users. The authors in [30] proposed a cloud–edge–client federated learning task prediction mechanism based on bidirectional long- and short-term memories, in which participants only use local training models without

uploading data to servers. This algorithm avoids solving complex optimization problems while ensuring user privacy and security. The authors of [31] considered the problem of conflict between resource consumption and quality of service and rationally allocate computational resources through a resource ranking method so as to reduce the probability of task failure. The authors of [32] designed an intelligent collaborative framework scheme to realize dynamic service prediction and multi-task offloading decisions. This scheme enables better task sensing, anomaly inference, and task offloading.

The task prediction offloading research has a significant effect on coping with the complexity of the offloading environment caused by the diversity of user behaviors, which is helpful in alleviating the environmental overload phenomenon and improving the problem of long system-response delays. However, given that task forecasting is closely related to resource allocation, the reservation of compute resources should be fully considered when making task forecasting. The authors in [33] proposed a delay-aware resource reservation strategy based on spatio-temporal and spatial-task decentralization demand forecasting and proposed a regional-edge server resource reservation algorithm based on the prediction model to minimize the latency of terminal tasks. The authors in [34] updated the upload strategy based on feedback from the environment and optimized resource allocation with the help of Salp's artificial bee colony algorithm to improve the performance of IoT devices. The authors of [35] discussed offline decision-making and resource allocation in MEC systems, especially optimization for time-varying wireless fading channels and edge computing capabilities. The authors of [36] optimized the edge offloading model based on the task prediction of the LSTM algorithm, the computing offloading strategy of mobile devices based on task prediction, and the task migration of the edge cloud scheduling scheme. The authors of [37] used LSTM to track the resource characteristics of IoT over time and, based on the deep deterministic policy-gradient algorithm to assist in online calculation offline decision-making and task migration, and resource allocation, effectively reduced task turnaround time and system energy consumption. The research content of the above papers involves the combination of task offloading and prediction technology and has achieved certain results, effectively improving the task offloading efficiency, but it is not combined with the complex scenario of the dynamic changes of multi-user concurrent resources and lacks the close integration of actual scenarios. Therefore, the follow-up research should focus on task offloading and prediction technology in the environment of multi-user concurrency and dynamic resource changes so as to better adapt to actual needs and further improve the performance and efficiency of the system.

In summary, in the smart IoT environment, it is still necessary to conduct in-depth research and improve the algorithms and mechanisms from multiple perspectives, such as timing dependency among tasks, dynamic changes of global resources, and node load in multi-user concurrent scenarios, to optimize the current application of mobile edge computing in the smart IoT. Therefore, this study focuses on the offloading of timing-dependent tasks in multi-user concurrent scenarios and examines collaborative offloading mechanisms among users, considering resource competition in dynamic environments as a starting point. It improves the learning efficiency of agent algorithms by combining prediction techniques with reinforcement learning, thereby reducing task offloading latency and improving the utilization of computing resources.

3. System Model

In the IoT multi-user concurrent scenario, the edge-computing task scheduling model is divided into two layers: the end-user layer and the edge service layer. As shown in Figure 1, in the end-user layer, user devices are randomly distributed and can access different edge servers within the sensing range, denoted as $U = \{u_1, u_2, \ldots, u_n\}$. Edge servers in the edge service layer are regularly distributed in the scenario, and an edge server can serve multiple end-users at the same time, denoted as $M = \{m_1, m_2, \ldots, m_n\}$. The service request applied by each end-user is composed of computation-intensive tasks with timing dependencies, so the number of internal tasks of a service request can be

denoted as $V = \{v_1, v_2, \ldots, v_n\}$. This system model aims to divide the service requests with timing-dependent tasks, so that different edge servers can operate simultaneously in multi-user concurrency scenarios, improving the response efficiency of multiple users as well as the resource utilization of the global edge servers. The reader is referred to Table A1 for the detailed parameters.

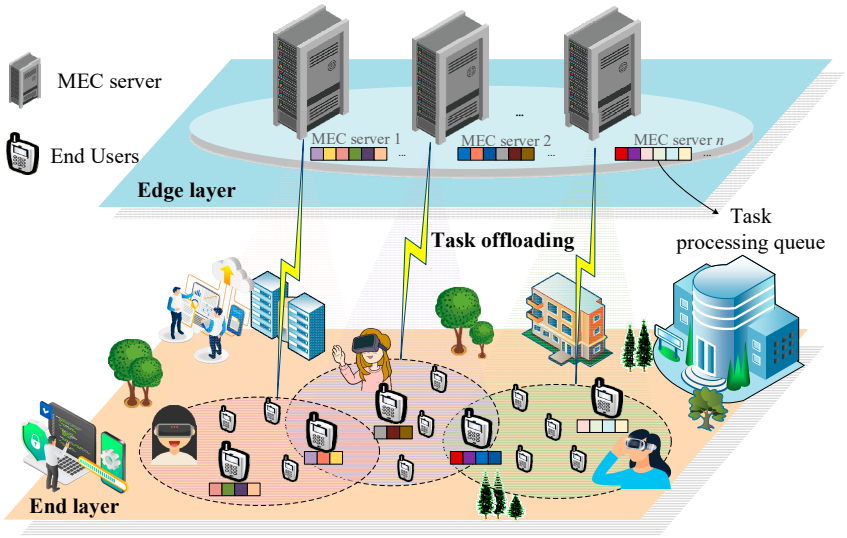


Figure 1. Timing-dependent task scheduling architecture.

3.1. Time-Dependent Task Model

The heterogeneity, networking, and complexity of applications in IoT scenarios make the task offloading problem increasingly complex. To improve the task response rate, abstract modeling becomes an important tool and the first step to solving this problem. In practical applications, because the timing dependency constraints between tasks need to be considered when responding to user service requests, a directed acyclic graph (DAG) can represent such dependencies more clearly. Therefore, existing studies usually use DAG to model tasks with timing dependency constraints.

In this paper, we use DAG G_d to denote the application program of the dth terminal device, $G_d = (V_d, E_{i,j})$, in which V_d denotes the set of subtasks in the graph and $E_{i,j}$ denotes the dependency between tasks if $E_{i,j} = 1$ denotes that the execution of task j requires the computation result of task i. On the contrary, if $E_{i,j} = 0$, it means that there is no dependency between task i and task j in terms of the order of sequential execution. For each subtask in the task set, V_d denoted as $V_i = (a_i, c_i, T_i^{tol})$, a_i is the size of the input data of task i, c_i is the number of CPU cycles required to compute each bit of data for the task, and T_i^{tol} is the maximum tolerable delay for computing the task.

Tasks in the DAG can choose whether to offload or execute locally based on their offloading delay and global resource utilization. If the task is offloaded to the MEC server, the offloading process is divided into three stages: sending data, executing the task, and returning the results. If the task is executed locally, the delay only needs to consider the time of local execution.

3.2. Task-Offloading Delay Evaluation Model

When adaptively scheduling tasks to appropriate task processing locations according to user service requests, service delays in two different situations need to be considered, including local execution or offloading to the MEC server for execution.

When the UE's service demand is executed locally, the UE does not need to transmit the task to the MEC server and can directly utilize the local computing resources for processing.

Therefore, for UE u_i and the task $v_{i,j} = (a_{i,j}, c_{i,j}, T_{i,j}^{tol})$ generated by u_i, the processing delay for u_i to complete task j locally is obtained as:

$$T_{i,j}^{loc} = \frac{a_{i,j}}{f_l}, \tag{1}$$

where f_l denotes the UE local computation rate.

When the local device cannot satisfy the service demand of the application program, the UE service request will be offloaded to the MEC server for execution. That is, if the local device is currently occupied by other programs that cannot provide the corresponding computing resources, or the processing time of the tasks required by the weak processing capability exceeds the deadline of the program response, the service request will be offloaded to the MEC server. At this time, the task goes through three stages: data transmission, task processing, and result return. In the data transmission stage and result return stage, the user transmits the task data to the corresponding MEC server through the channel. The uplink data transmission rate, downlink data transmission rate, uplink data transmission delay, and downlink data transmission delay can be calculated by Equations (2)–(5) [19]:

$$R_{d,n}^{ul} = B_{d,n}^{ul} \log_2(1 + \frac{P_{d,n}}{\eta^2}), \tag{2}$$

$$R_{d,n}^{dl} = B_{d,n}^{dl} \log_2(1 + \frac{P_{d,n}}{\eta^2}), \tag{3}$$

$$T_{i,j}^{up} = \frac{a_{i,j}}{R_{d,n}^{ul}}, \tag{4}$$

$$T_{i,j}^{down} = \frac{a_{i,j}}{R_{d,n}^{dl}}, \tag{5}$$

where $B_{d,n}^{ul}$ and $B_{d,n}^{dl}$ represent the communication channel bandwidth of uplink and downlink, respectively. $P_{d,n}$ represents the uplink/downlink channel transmission power. η^2 represents the channel noise during the transmission process.

The calculation delay of processing tasks on the MEC server can be calculated by Equation (6):

$$T_{i,j}^{n\_cal} = \frac{a_{i,j} c_{i,j}}{f_n}, \tag{6}$$

where f_n represents the calculation rate of the MEC server.

Therefore, if the service requirements of UE are offloaded to the MEC server for execution, the resulting task processing delay can be calculated by Equation (7):

$$T_{i,j}^{sum} = T_{i,j}^{up} + T_{i,j}^{n\_cal} + T_{i,j}^{down}. \tag{7}$$

3.3. Computing Resource Forecasting Model

The GRU combines parameter information such as historical offloading decisions, task attributes, and computational resources and utilizes historical information about the end device to predict the task that arrives at the next state of the end-device network. The GRU controls the flow of information through the introduction of update gates and reset gates, and it performs better in capturing long-range dependencies in a time series. Update gates help the model to decide how much previous information to keep in the current hidden state. Reset gates are used to decide which historical data to discard. The structure of these gates allows the GRU to update information more efficiently at each time step, so GRU-based models are trained faster. The complete GRU update process is denoted by Equations (8)–(11) [38]:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]), \tag{8}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]), \tag{9}$$

$$\widetilde{h_t} = \tanh(W \cdot [r_t \times h_{t-1}, x_t]), \tag{10}$$

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times \widetilde{h_t}, \tag{11}$$

where x_t denotes the input vector, h_{t-1} denotes the state memory variable at the previous moment, h_t denotes the state memory variable at the current moment, z_t denotes the update gate state, r_t denotes the reset gate state, $\widetilde{h_t}$ denotes the state of the current candidate set, σ denotes the sigmoid activation function, and W denotes the weight parameter for the multiplication of the connection matrix.

In the MEC network scenario, this paper predicts information about future tasks through the GRU model to make decisions and allocate computing resources to tasks. When the actual task arrives, it is offloaded and recomputed if the error range between the actual task and the predicted task is within the allowed threshold. The accuracy of the prediction model is improved by training the GRU network model and updating the weights and biases of the gates in the network.

3.4. Problem Formulation

In realistic IoT environments, each node has its fixed computing resource which forms the basis for supporting responses to UE service requests. The main goal of this paper is to efficiently utilize the computing resources of the MEC to minimize the total offloading delay of global IoT device service requests. An additional goal is maximizing the node resource utilization to optimize the response efficiency of the system in handling concurrent service requests from multiple IoT devices per unit of time. To achieve this goal, this paper establishes the objective function and constraints denoted by Equation (12). This means that the execution location of a task is determined by the offloading delay of the equipment to be processed for that task and the current resource utilization of the equipment to be processed. The offloading delay and resource utilization are combined and weighed to determine the optimal offloading location for this task. The total offload delay and the resource utilization are defined as Equations (13) and (14), respectively:

$$w_1 \min T_{total} + w_2 \max U_{total}, w_1 + w_2 = 1\,, \tag{12}$$

$$T_{total} = \sum_{i=1}^{|Q|} \max_{j=1}^{|Q_i|} T_{i,j} + \sum_{i \in F} t_i,\ t_i = \begin{cases} 0\,, & if\ i\ is\ in\ the\ parallel\ set \\ t_i, & otherwise \end{cases}\,, \tag{13}$$

$$U_{total} = \sum_{j=1}^{J} \alpha_{j,k}^{util} = \sum_{j=1}^{J} \frac{\alpha_j^{util}}{\alpha_k^{util\_\max}}\,, \tag{14}$$

$$\begin{aligned} s.t. C1 &: 0 < T_{d,j}^{sum} \le T_{d,j}^{tol} \\ C2 &: T_{j\_finish} \le T_{j+1\_start}, 1 \le i \le I, 1 \le j, j+1 \le J \\ C3 &: x_{k,j}^{n} + x_{k,j}^{d} = 1, 1 \le k \le K, 1 \le j \le J \\ C4 &: f_L \le f_L^{\max}, f_N \le f_N^{\max} \\ C5 &: P_{d,m} + P_{d,n} \le P^{\max} \\ C6 &: \alpha_{j,n}^{util} \le \alpha_{j,n}^{util\_\max}, \forall n \in N \end{aligned}$$

where F denotes the set of all serial tasks, Q denotes the set of all parallel tasks, Q_i denotes the ith subset of the parallel set, $T_{i,j}$ denotes the ith subtask latency in the jth parallel set, and t_i denotes the execution latency of the ith serial task. $\alpha_{j,k}^{util}$ denotes the resource utilization when offloading task j to node k. $\alpha_k^{util\_max}$ denotes the maximum available computing resource on node k, and α_j^{util} denotes the computing resource required for task j.

Equation (13) indicates that the offloading delay of a dependency request service is equal to the sum of the offloading delay of the longest task in all parallel sets and the offloading

delay of all serial tasks. Constraint *C*1 indicates that the offloading delay of each task is less than its maximum tolerable delay. Constraint *C*2 indicates that the precondition for task j to start offloading is that all of its predecessors with dependencies have finished executing. Constraint *C*3 indicates that the task has only the possibility of being executed on one device, which includes the UE and the MEC server. Constraint *C*4 is a power constraint for the UE and the MEC server. Constraint *C*5 indicates that the transmission power of the UE d can guarantee the normal offloading of the application tasks. Constraint *C*6 indicates that the remaining computing resources of the edge server are fully available when the tasks are offloaded.

4. User-Prediction-Based Timing-Dependent Task Offloading Algorithm

For the problem of task offloading in multi-user concurrent scenarios, this section first parses the tasks and obtains the serial and parallel task scheduling list for multi-server parallel processing with respect to the characteristic attributes of the timing-dependent tasks. Then, the environment configurations in the MADDPG algorithm and the improvement strategies of the algorithm are described with the goal of obtaining the optimal offloading locations and indexes of each subtask in the global state.

4.1. Timing-Dependent Task Preprocessing

In realistic IoT scenarios, each terminal generates independent service requests that contain tasks related to the chronological order. That is, certain subtask executions require the output of the previous subtask as input. If not handled properly, it may affect the service response time and reduce the QoE of users. Therefore, this paper proposes the DFS-based, timing-dependent task preprocessing (DTP) algorithm to generate a serial and parallel scheduling list of timing-dependent tasks, which lays the foundation for subsequent task offloading. In view of the DFS algorithm as a basic graph traversal algorithm, its detailed working principle can be found in the literature [39].

Figure 2 shows the task-offloading processing when two users apply for different services. In the multi-task concurrency scenario, subtasks in different applications, as well as subtasks in the same application without timing dependency constraints, can be processed in parallel while satisfying the offloading requirements.

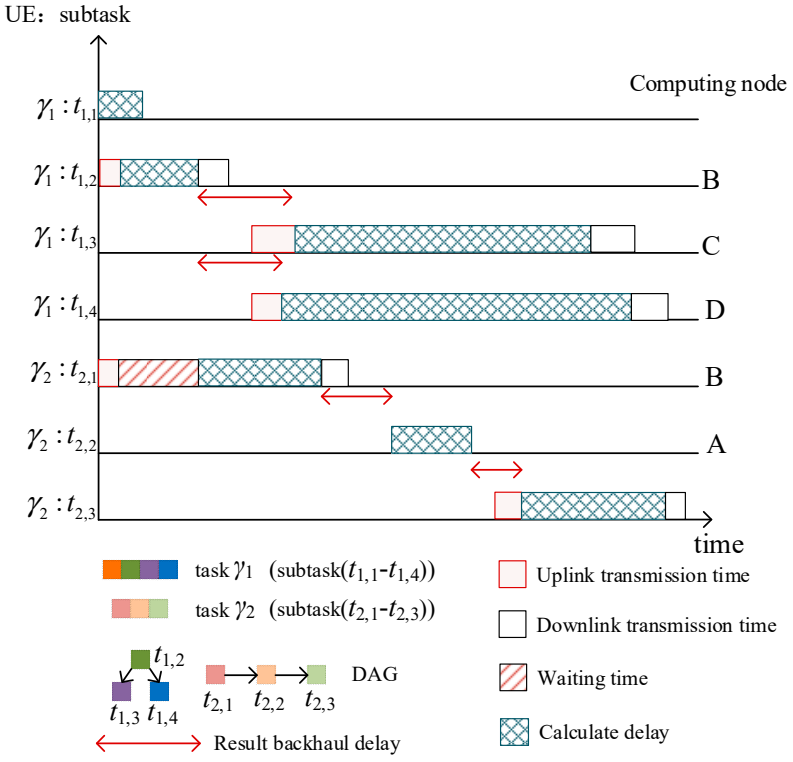


Figure 2. Task scheduling diagram for a multi-user concurrency scenario.

4.2. Timing-Dependent Task Offloading Algorithm

MADDPG is applicable to a variety of complex environments and systems that require multiple agents to collaborate or compete to accomplish a task. The Markov decision of N agents is defined by a set of states $S = [o_1, o_2, \ldots, o_n]$ that describe the independently observed states of all the agents themselves, and the agents choose an action from action $A = [a_1, a_2, \ldots, a_n]$ as feedback to the current state. Agents interact with the environment, and every agent that makes a corresponding action causes a corresponding change in the state environment. Therefore, the environment is non-stationary from the point of view of any agent.

The core idea of the MADDPG algorithm lies in learning a centralized Q-function that is conditional on global information for each agent, aiming to alleviate the non-stationary problem and promote the stability of the training process [40]. In addition, to reduce the variance of the policy gradient, centralized training is adopted in an actor-critic learning framework [41]. The system architecture of centralized training is shown in Figure 3. During the training process, the central controller randomly extracts a batch of records (o_t, a_t, r_t, o_{t+1}) from the experience playback cache to train the agent's policy network parameters. The value network is used to evaluate the value of the agent taking a specific action a in a specific state s_t. The expected gain gradient for agent i is [42]:

$$\nabla_{\theta_i} J(\mu_i) = E_{x\sim D}[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i} Q_i^{\mu}(x, a_1, \ldots, a_n)|_{a_i=\mu_i(s_i)}], \tag{15}$$

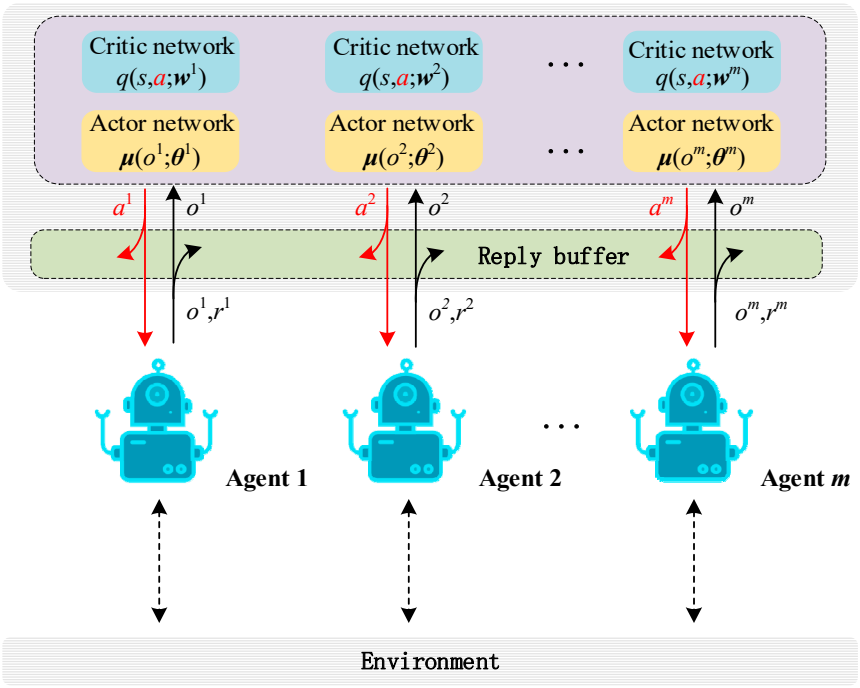


Figure 3. The training process based on the MADDPG task-offloading algorithm.

The loss function for training Q is defined as [39]:

$$L(w_i) = E_{x,a,r,x'}[(Q_i^{\mu}(x, a_1, \ldots, a_n) - y)^2], \tag{16}$$

$$y = r_i + \gamma Q_i^{\mu'}(x', a'_1, \ldots, a'_n)|_{a'_j=\mu'_j(o_j)}, \tag{17}$$

After the training is completed, the policy network parameters are deployed to the corresponding agents who can make decisions locally and independently based on the local observation o and no longer need the participation of the value network. The training flow based on the MADDPG task-offloading algorithm is shown in Figure 3.

In order to solve the computing resource conflicts in multi-task concurrent scenarios, this paper improves the training process of the MADDPG algorithm to obtain better and faster offloading strategies. It is assumed that at the moment t, GRU-based deep reinforcement learning obtains the state of the server in the MEC network from the existing multi-user concurrent scenario and makes the corresponding offloading action a_t based on the observation value. The offloading action directly affects the current service user state and updates the existing task state, and the user decides whether to offload the task or not based on the action value. When the task enters the offloading state, it will go through different stages of state changes such as data transmission, edge node storage queuing, node processing, and execution completion. During the task execution, the computing resources of the MEC server continuously and dynamically change. After the execution of this round of action, the agent obtains the corresponding feedback r_t from the environment and the state S_{t+1} of the environment at the next moment. The agent stores the data (s_t,a_t,r_t,s_{t+1}) recorded in this round of training in the return experience pool and utilizes the data to train its own network data. As can be seen in Figure 4, this paper improves the training process of reinforcement learning, in which the agent cannot obtain the immediate reward r_t from the environment feedback but only the reward r_{t-1} obtained from the training in the previous moment. At the moment t+1, because the GRU module in the deep neural network helps to memorize the historical offloading decisions and the task information, the agent selects the current offloading action based on the information of the user's behavioral prediction, the reward r_t from the previous moment, and the current moment t+1 observation information, to select the offloading action for the current round.

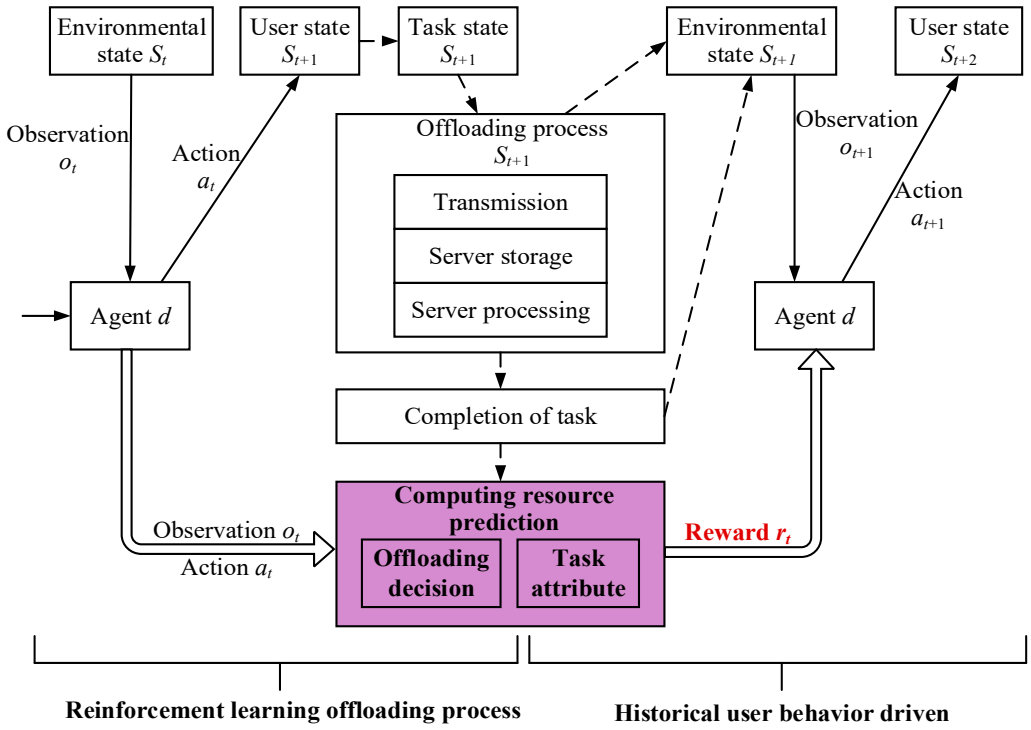


Figure 4. Training process based on the UBP-DTO algorithm.

In complex scenarios with multiple concurrent users, task offloading using multi-agent reinforcement learning has the problem of poor timeliness of the offloading strategy due to its long training time. Therefore, the user-behavior-prediction-based dependent task offloading (UBP_DTO) algorithm is proposed. The proposed algorithm considers historical offloading decision information and task resource occupancy to predict the computing resource usage of the task at the next moment, improving the training efficiency of reinforcement learning and the timeliness of offloading strategy. The specific optimization process is shown in Algorithm 1.

The main content of the UBP_DTO algorithm optimization is shown in line 9 of Algorithm 1, which changes the mechanism for obtaining instant rewards during the reinforcement learning training process. When training and selecting actions, the agent focuses on historical user behaviour information and selects actions based on effective task offloading decisions in historical information, the current environment computing resource status, and the CPU resources required for precursor tasks, narrowing the action selection range and improving training efficiency.

Algorithm 1: MADDPG-based timing-dependent task offloading algorithm
Input: *Task serial and parallel scheduling list SL.*
Output: *Independent offloading strategy* πd *for each agent d.*

1: Initialize the environment, including the resources, quantity, and location of MEC nodes and users
2: Agent d observes the state of the environment o_t and selects the action a_t
3: Obtain the environment state feedback reward r_t
4: **for** episode = 2 to max_episode **do**
5: Reset the MEC network
6: Get the task scheduling sequence
7: **while** the episode do not end **do**
8: **for** agent d in the set of agents **do**
9: Observe the o_{t+1}, r_t and select the action $a_{t+1} = \mu_d(o_{t+1}, r_t; \theta_d)$
10: **end for**
11: Executing a_{t+1}:(o_{t+1}, r) = *Actions*$[a_d]$ in parallel
12: **for** agent d in the set of agents **do**
13: Record the experience $(o_t^d+1, a_t^d+1, r_t^d+1, o_t^d+2)$ of agent d and put it into the experience replay pool
14: **end for**
15: **for** agent d in the set of agents **do**
16: Update policy network (agent d)
17: Update value network (agent d)
18: **end for**
19: $\theta'_d \leftarrow \tau\theta_d + (1-\tau)\theta'_d$
20: **end while**
21: **end for**
22: Each agent autonomously decides the task offloading mode and generates the offloading policy π_d
23: Return π

In a multi-user concurrent-timing-dependent task offloading system, each UE is configured with an independent agent that can initiate service requests at the same time. All user terminal agents need to make task offloading and location selection decisions based on changing environmental information. For this process, this article makes the following environment configuration for the edge-computing task offloading solution based on the MADDPG algorithm:

State: x_t denotes the distributed state o_t observed by all local UEs at moment t. The overall environmental state can be denoted as $o_t = \{Z_n, R_n, T_{sum}, RL, D_j\}$. Z_n denotes the location of the MEC node observed by each agent, which is used to compute the transmission time for task offloading. R_n denotes the computing resources of the MEC node observed by each agent and can be used to determine whether the node has sufficient computing resource to support task offloading. It can also be used to evaluate the potential benefit of the current task offloading. T_{sum} denotes the task offloading time. RL is a preprocessed task-scheduling sequence, in which the correct ordering among the tasks guarantees the success rate of the agent's task offloading. D_j denotes the size of the data of the current computational task j and the size of its return data.

Action: Use a_t to represent actions and define action sets as agents that issue service requests globally, as well as optional task-offloading locations, including local UE and

MEC servers. Evaluate the effectiveness of different offloading strategies by calculating the overall task offloading delay and learn the optimal action plan to minimize the total offloading delay while maximizing the resource utilization of global nodes.

Reward: A reward function is a mechanism for measuring behavioral performance in a given environment. The agent uses the immediate reward value to evaluate the effectiveness of its actions, thereby prompting it to learn the optimal task-offloading strategy. This paper considers the comprehensive impact of the time overhead of task offloading and the resource utilization efficiency in the process to define the reward r. The agents use the reward values from the previous step to evaluate the effectiveness of their actions, and after weighing the summed benefits of the current task offloading latency and server resource utilization, the offloading strategy with the optimal value of the summed benefits is used as the learning objective to train all the agents' network parameters. The specific expressions of the reward function and resource utilization as shown in Equations (18) and (19), respectively:

$$r(j,k) = w_1 * T_{d,j}^{sum\_inv} + w_2 * \alpha_{j,k}^{util} \ , \ w_1 + w_2 = 1 \, , \tag{18}$$

$$\alpha_{j,k}^{util} = \frac{\alpha_j^{util}}{\alpha_k^{util\_\max}} \, , \tag{19}$$

where w_1, w_2 denotes the weights in the definition of the reward function. $T_{d,j}^{sum\_inv}$ denotes the reciprocal of the processing delay of the jth task of the dth application.

If a task is offloaded to one of the edge nodes, the offloading process includes transmission latency, execution latency, and result return latency. In the case of local task processing, only the local execution delay is considered. In order to allocate enough space for computing resources to the remaining tasks, the reward function aims to guide the agents to prioritize the offloading of tasks to nodes with less available resources.

5. Experimental Results and Discussion

5.1. Parameter Settings

In this paper, we design the locations of user devices and MEC servers in a 3D Cartesian coordinate system. Specifically, we place the MEC nodes at coordinates (1,2,0), (2,1,0), and (2,2,0.1) and set the radius of their service ranges to 1.5 km. Meanwhile, the ground agents are deployed at (1,1,0), (2,3,0), and (3,1,0), and the perception radius of each agent is similarly set to 1.5 km. Such a simulation scenario design not only realizes the offloading process between a single agent and a single server but also simulates a dynamic and complex environment in which a single server serves multiple agents simultaneously, in order to verify the applicability of the proposed algorithm.

To enhance the realism of the experiments, this paper sets up an agent to randomly retrieve application offloading information from the dataset, in which each DAG exhibits a unique structure of task timing dependencies in the application. However, due to the limited application data in the current real dataset, it is difficult to meet the demand for offloading model training [43]. Therefore, in this paper, we adopt the same approach as in the literature [44] and utilize a synthetic DAG generator [43] to create a variety of DAGs representing heterogeneous applications. The complexity of the DAG is controlled by factors such as density and dimensionality, in which the density denotes the number of inter-task edges in the DAG, and the dimensionality determines the width and height of the graph. It is worth noting that each user's service request contains a varying number of subtasks, and even for DAG with the same number of subtasks, the subtasks still have different timing dependencies among them. Because many applications in IoT scenarios are computation-intensive, the DAG is generated by setting randomly selected parameters from {0.3,0.4,0.5,0.6,0.7,0.8} to control the dimension and density of the DAG. In addition, in this paper, the data size of each subtask is set to 640–6400 kb. In this paper, we refer to the literature [19,45] to set the simulation parameters, as shown in Table 1.

Table 1. Simulation parameters.

Parameters	Value
Location and service radius of edge nodes	(1,2,0) (2,1,0) (2,2,0.1) 1.5 km
Location and sensing radius of the agent	(1,1,0) (3,1,0) (2,3,0) 1.5 km
Terminal CPU frequency	2×10^{10} cycles/s
CPU frequency of the edge node	8×10^{11} cycles/s
Task computing resource allocation ratio	0.27 × task size bit
Total computing resources of the terminal	10 MB
Total computing resources of the edge nodes	35 MB
Uplink/downlink transmission power of edge nodes	35 W
Transmission bandwidth of edge nodes	10 MHz
Task data size	[640, 6400] KB

To evaluate the scheme proposed in this paper, five existing task offloading solutions are compared as follows:

(1) Remote offloading algorithm: all tasks in the DAG are offloaded to the MEC server on the ground for execution.
(2) Greedy algorithm: each task in the DAG is greedily assigned to either local or remote execution based on the algorithm's estimation of the local processor's or the MEC node's completion time.
(3) Deep Q-network-based dependent task offloading algorithm (DQN_DTO): using deep Q-networks, the agent automatically learns how to choose the optimal task offloading decision under different environment states by learning the reward signals in the environment.
(4) DDQN-based dependent task-offloading algorithm (DDQN _DTO): agents are trained by alternating between these two neural networks to dynamically adjust and optimize the task offloading strategy to achieve a more accurate benefit trade-off.
(5) MADDPG-based dependent task-offloading algorithm (MADDPG_DTO): By continuously training the MADDPG algorithm, multiple agents collaborate with each other to jointly weigh the offloading benefits and generate offloading strategies.

5.2. Parameter Settings

In this section, the proposed DTP algorithm is combined with two DRL algorithms, DQN and DDQN, respectively, to verify the fitness and stability of the DTP algorithm. As shown in Figure 5, the rewards of both DQN_DTP_TO and DDQN_DTP_TO offloading algorithms combined with the DTP algorithm grow steadily as the training ground continues to be iterated, and both converge after nearly 650 rounds of training. Figure 5 compares the DQN_DTP_TO algorithm and the DDQN_DTP_TO algorithm with the sequential offloading based on DQN and DDQN, respectively, and the training results show that as the learning rounds continue to increase, the response latency of the offloading algorithms in combination with the DTP algorithms is lower than that of the baseline algorithm, and the offloading latency is improved by nearly 6.9% in terms of reducing the offloading latency.

The proposed DTP algorithm is suitable for deep reinforcement-learning offloading algorithms, maintains stability in DRL algorithm training, and improves better adaptation in terms of task-offloading efficiency. Therefore, in MEC network scenarios, the rational use of hierarchical dependencies between different tasks and the resources of multiple edge nodes can effectively improve the response rates of end-users, and the proposed DTP algorithm lays the foundation for the subsequent processing and offloading tasks in more complex application scenarios.

Figure 6 compares the convergence performance of the proposed UBP_DTO with the other DRL offloading algorithms. As shown in Figure 6, under the same conditions, the proposed UBP_DTO algorithm converges after nearly 500 Episode rounds, and not only converges faster but also the reward value is significantly better than the other three compared algorithms. This is due to the fact that the proposed algorithm utilizes the GRU

module to record the feature attributes of the timing-dependent tasks and the historical offloading decisions to help the agent predict the attribute information of the next upcoming task. It reduces the complexity of the scenarios in the successive decision-making environment on the basis of the MADDPG offloading algorithm and thus improves the offloading decision-making efficiency. The combination of MADDPG and GRU enables the agent not only to make the optimal decision based on the current state but also to better predict the future state based on the historical behavior and the trend of the environment. Meanwhile, the GRU module can help the algorithm better adapt to the dynamically unstable environment when it needs to consider other agent strategies and environment states. Therefore, compared with the single-agent algorithms, the proposed algorithm obtains higher rewards in the early stage of learning. It effectively illustrates the effectiveness of the offloading decisions made by the MADDPG-based multi-agent offloading algorithm compared to the single-agent offloading algorithm even in the early stage of training.

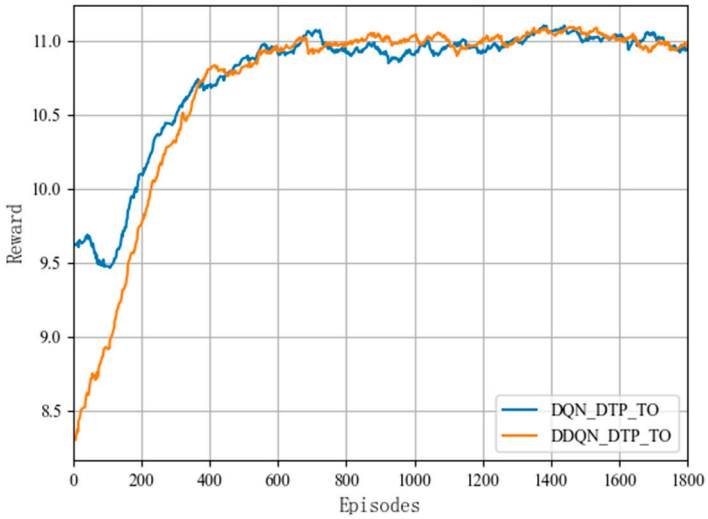
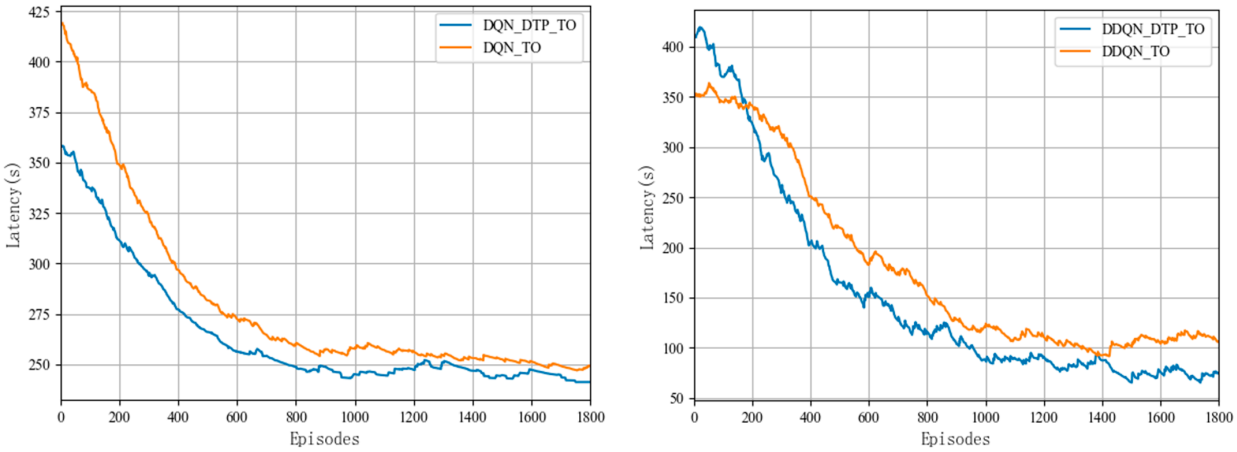


Figure 5. Fitness performance of two DRL algorithms combined with the DTP algorithm.



(**a**) (**b**)

Figure 6. Comparison of the delay of the DTP algorithm combined with two DRL algorithms: (**a**) DQN and (**b**) DDQN.

In addition, as shown in Figure 7 about the convergence details of the two algorithms MADDPG_DTO and UBP_DTO, compared with MADDPG_DTO algorithm, UBP_DTO algorithm short 100 episodes and have a higher reward convergence value. Therefore, the GRU-MADDPG-based task offloading scheme effectively improves the training rate and training effect of MADDPG.

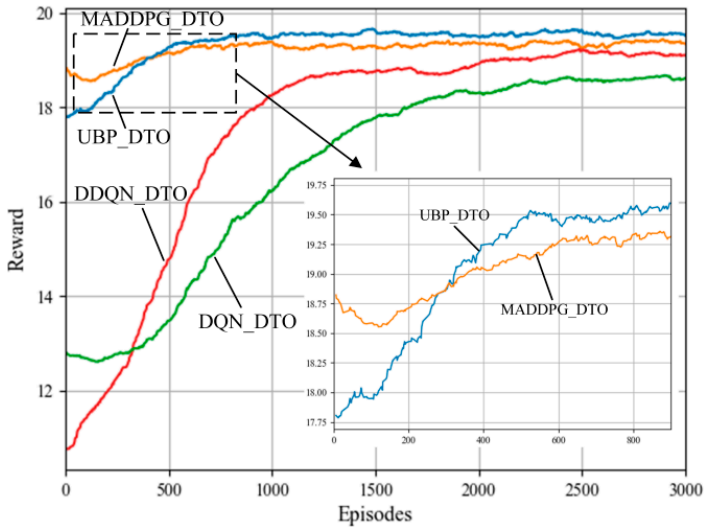


Figure 7. Convergence results of different reinforcement learning algorithms.

Figure 8 demonstrates the performance comparison of the proposed algorithm with five other benchmark algorithms in terms of offloading delay. As shown in Figure 8, the MADDPG_DTO algorithm and the UBP_DTO algorithm based on MADDPG show good performances in scenarios in which the computing resources dynamically change. According to the offloading delay results, the UBP_DTO algorithm based on user behaviour prediction outperforms the MADDPG_DTO algorithm in terms of offloading delay by nearly 6.7% and is higher than other benchmark algorithms. Meanwhile, Figure 9 demonstrates the performance comparison between the proposed algorithm and five other benchmark algorithms in terms of resource utilization under different physical locations, even though the proposed algorithm does not perform optimally locally. However, the proposed UBP_DTO algorithm maintains a smooth resource utilization in the vast majority of cases and performs optimally on average, with an improvement of 30.6% compared to the suboptimal algorithm, when observed from the perspective of comprehensive global resource utilization. This is due to the fact that the agent considers the strategies of other agents on the basis of historical offloading decisions and task attribute information, which maximizes the accuracy of the offloading action from the perspective of the global offloading interests of all agents and effectively utilizes the global computing resources.

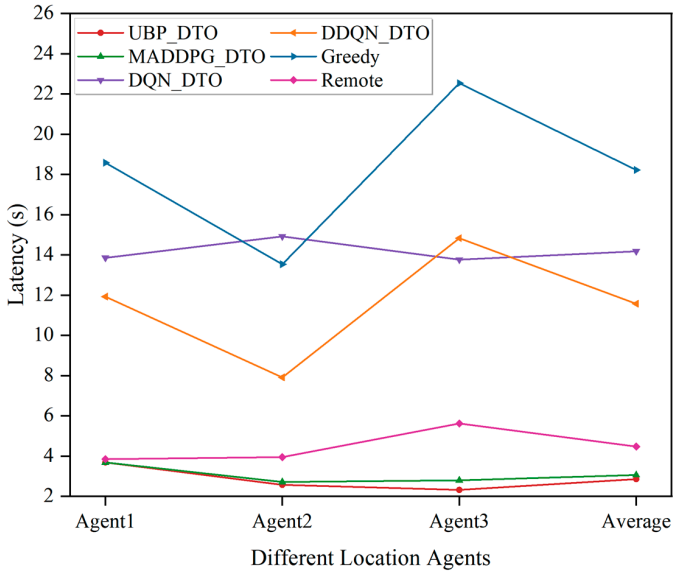


Figure 8. Comparison of agent offloading delay in different physical locations.

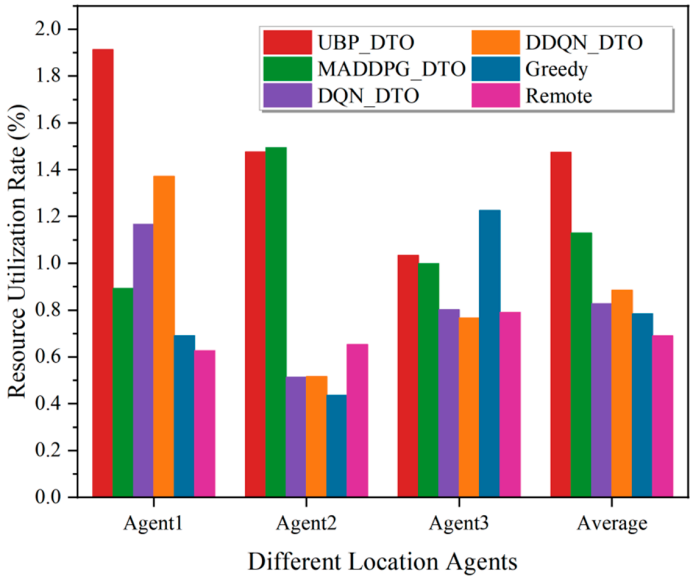


Figure 9. Comparison of agent resource utilization at different physical locations.

6. Conclusions

With the widespread popularization of smart IoT technology and the sharp increase in the number of timing-dependent applications, the importance of user experience quality is becoming more and more prominent, which in turn puts forward more stringent requirements on system performance, response speed, and accuracy of task processing. In this paper, we design an edge-computing, timing-dependent task scheduling system to improve the response speed of users and the resource utilization of MEC servers in multi-user concurrent scenarios. To solve the task-timing dependency constraints and resource conflicts of the multi-user concurrent application services, the GRU is used to memorize the effective offloading information and predict the subsequent resource demand. Meanwhile, the training process of the multi-agent offloading algorithm is improved, and the global benefit is enhanced based on historical experience and collaboration among multi-agents. The simulation results show that the proposed algorithm effectively overcomes the problems of over-fitting in the learning process of single-agent algorithms and the difficulty of a single local decision to comprehensively optimize the global benefit, and it improves the global resource utilization rate, user response efficiency and timeliness of the offloading strategy. However, this study addresses the under-explored security and privacy aspects of data in the context of multi-agent collaboration, and we will focus on this to further improve our work in the future.

Author Contributions: Conceptualization, Y.W. and J.Z.; methodology, Y.W.; software, Y.W.; validation, Y.W.; formal analysis, Y.W. and Z.H.; investigation, Y.W. and Z.H.; resources, Y.W.; data curation, Y.W.; writing—original draft preparation, Y.W.; writing—review and editing, Y.W. and Z.H.; visualization, Y.W. and Z.H.; supervision, Z.W.; project administration, Z.W.; funding acquisition, J.Z, Z.W. and Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Hebei Province Innovation Capacity Enhancement Program Project under Grant No. 22567624H, the Handan Science and Technology Research and Development Program under Grant No. 21422031288 and the Provincial Innovation Funding Project for Graduate Students of Hebei Province under Grant (No. CXZZSS2023120).

Data Availability Statement: The data in this paper are not publicly available at this time.

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

Appendix A

Table A1. Notation table.

Symbol	Description
G_d	the application program of the dth terminal device
V_d	the set of subtasks in the graph
E_i	the dependency between tasks
a_i	the size of the input data of task i
c_i	the number of CPU cycles required to compute each bit of data for the task
T_i^{tol}	the maximum tolerable delay for computing the task
f_l	the UE local computation rate
f_n	the calculation rate of the MEC server
$B_{d,n}^{ul}$,	the communication channel bandwidth of the uplink
$B_{d,n}^{dl}$	the communication channel bandwidth of the downlink
$P_{d,n}$	the uplink/downlink channel transmission power
η^2	the channel noise during the transmission process
$R_{d,n}^{up}$	the uplink data transmission rate
$R_{d,n}^{dl}$	the downlink data transmission rate
$T_{i,j}^{up}$	the uplink data transmission delay
$T_{i,j}^{down}$	the downlink data transmission delay
$T_{i,j}^{n\_cal}$	the calculation delay of processing tasks on the MEC server
x_t	the input vector
h_t	the state memory variable
h_{t-1}	the state memory variable at the previous moment
$\tilde{h}_t$	the state of the current candidate set
σ	the sigmoid activation function
W	the weight parameter for the multiplication of the connection matrix
F, Q, Q_i	the set of all serial tasks, the set of all parallel tasks, the ith subset of the parallel set
$T_{i,j}$	the ith subtask latency in the jth parallel set
t_i	the execution latency of the ith serial task
T_{total}	the total offload delay
U_{total}	the resource utilization
w_1, w_2	the weights in the definition of the reward function
$T_{d,j}^{sum\_inv}$	the reciprocal of the processing delay of the jth task of the dth application
$\alpha_{j,k}^{util}$	the resource utilization when offloading task j to node k
$\alpha_k^{util\_max}$	the maximum available computing resource on node k
α_j^{util}	the computing resource required for task j

References

1. Aikyn, N.; Zhanegizov, A.; Aidarov, T.; Bui, D.; Tu, N.A. Efficient facial expression recognition framework based on edge computing. *J. Supercomput.* **2024**, *80*, 1935–1972. [CrossRef]
2. Gong, T.; Zhu, L.; Yu, F.R.; Tang, T. Train-to-Edge Cooperative Intelligence for Obstacle Intrusion Detection in Rail Transit. *IEEE Trans. Veh. Technol.* **2024**, 1–13. [CrossRef]
3. Wang, M.; Zhang, Y.; He, X.; Yu, S. Joint scheduling and offloading of computational tasks with time dependency under edge computing networks. *Simul. Model. Pract. Theory* **2023**, *129*, 102824. [CrossRef]
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. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [CrossRef]
6. Yang, J.; Yuan, Q.; Chen, S.; He, H.; Jiang, X.; Tan, X. Cooperative task offloading for mobile edge computing based on multi-agent deep reinforcement learning. *IEEE Trans. Netw. Serv. Manag.* **2023**, *20*, 3205–3219. [CrossRef]
7. An, X.; Fan, R.; Hu, H.; Zhang, N.; Atapattu, S.; Tsiftsis, T.A. Joint task offloading and resource allocation for IoT edge computing with sequential task dependency. *IEEE Internet Things J.* **2022**, *9*, 16546–16561. [CrossRef]
8. Wang, J.; Hu, J.; Min, G.; Zomaya, A.Y.; Georgalas, N. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 242–253. [CrossRef]

9. Gao, S.; Wang, Y.; Feng, N.; Wei, Z.; Zhao, J. Deep reinforcement learning-based video offloading and resource allocation in noma-enabled networks. *Future Internet* **2023**, *15*, 184. [CrossRef]
10. Ju, Y.; Chen, Y.; Cao, Z.; Liu, L.; Pei, Q.; Xiao, M.; Ota, K.; Dong, M.; Leung, V.C. Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 5555–5569. [CrossRef]
11. Guo, Y.; Xu, X.; Xiao, F. MADRLOM: A Computation Offloading Mechanism for Software-Defined Cloud-Edge Computing Power Network. *Comput. Netw.* **2024**, *245*, 110352. [CrossRef]
12. Wang, Q.; Wang, Q.; Zhao, H.; Zhang, H.; Zhu, H.; Wang, X. Device-Specific QoE Enhancement Through Joint Communication and Computation Resource Scheduling in Edge-Assisted IoT Systems. *IEEE Internet Things J.* **2024**, *11*, 13257–13270. [CrossRef]
13. Maray, M.; Mustafa, E.; Shuja, J.; Bilal, M. Dependent task offloading with deadline-aware scheduling in mobile edge networks. *Internet Things* **2023**, *23*, 100868. [CrossRef]
14. Tang, T.; Li, C.; Liu, F. Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning. *Comput. Commun.* **2023**, *209*, 78–90. [CrossRef]
15. Nguyen, L.X.; Tun, Y.K.; Dang, T.N.; Park, Y.M.; Han, Z.; Hong, C.S. Dependency tasks offloading and communication resource allocation in collaborative UAVs networks: A meta-heuristic approach. *IEEE Internet Things J.* **2023**, *10*, 9062–9076. [CrossRef]
16. Liu, J.; Ren, J.; Zhang, Y.; Peng, X.; Zhang, Y.; Yang, Y. Efficient dependent task offloading for multiple applications in MEC-cloud system. *IEEE Trans. Mob. Comput.* **2021**, *22*, 2147–2162. [CrossRef]
17. Liu, S.; Yu, Y.; Lian, X.; Feng, Y.; She, C.; Yeoh, P.L.; Guo, L.; Vucetic, B.; Li, Y. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 538–554. [CrossRef]
18. Al-hammadi, I.; Li, M.; Islam, S.M.; Al-Mosharea, E. A hierarchical hybrid subtask scheduling algorithm in UAV-assisted MEC emergency network. *IEEE Internet Things J.* **2021**, *9*, 12737–12753.
19. Zhang, X.; Lin, T.; Lin, C.K.; Chen, Z.; Cheng, H. Computational Task Offloading Algorithm Based on Deep Reinforcement Learning and Multi-Task Dependency. *Theor. Comput. Sci.* **2024**, *993*, 114462. [CrossRef]
20. Hosny, K.M.; Awad, A.I.; Khashaba, M.M.; Fouda, M.M.; Guizani, M.; Mohamed, E.R. Enhanced multi-objective gorilla troops optimizer for real-time multi-user dependent tasks offloading in edge-cloud computing. *J. Netw. Comput. Appl.* **2023**, *218*, 103702. [CrossRef]
21. Zeng, C.; Wang, X.; Zeng, R.; Li, Y.; Shi, J.; Huang, M. Joint optimization of multi-dimensional resource allocation and task offloading for QoE enhancement in Cloud-Edge-End collaboration. *Future Gener. Comput. Syst.* **2024**, *155*, 121–131. [CrossRef]
22. Sun, Y.; Li, H.; Wei, T.; Zhang, Y.; Wang, Z.; Wu, W.; Fang, C. Dependency-aware flexible computation offloading and task scheduling for multi-access edge computing networks. In Proceedings of the 2021 24th International Symposium on Wireless Personal Multimedia Communications (WPMC), Okayama, Japan, 14–16 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
23. Shu, C.; Zhao, Z.; Han, Y.; Min, G.; Duan, H. Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach. *IEEE Internet Things J.* **2019**, *7*, 1678–1689. [CrossRef]
24. Hou, W.; Wen, H.; Song, H.; Lei, W.; Zhang, W. Multiagent deep reinforcement learning for task offloading and resource allocation in cybertwin-based networks. *IEEE Internet Things J.* **2021**, *8*, 16256–16268. [CrossRef]
25. Kang, H.; Chang, X.; Mišić, J.; Mišić, V.B.; Fan, J.; Liu, Y. Cooperative UAV resource allocation and task offloading in hierarchical aerial computing systems: A MAPPO based approach. *IEEE Internet Things J.* **2023**, *10*, 10497–10509. [CrossRef]
26. Song, S.; Ma, S.; Zhao, J.; Yang, F.; Zhai, L. Cost-efficient multi-service task offloading scheduling for mobile edge computing. *Appl. Intell.* **2022**, *52*, 4028–4040. [CrossRef]
27. Chen, S.; Wang, X.; Sun, Y. TODO: Task offloading decision optimizer for the efficient provision of offloading schemes. *Pervasive Mob. Comput.* **2024**, *99*, 101892. [CrossRef]
28. Zaman, S.K.U.; Jehangiri, A.I.; Maqsood, T.; Haq, N.U.; Umar, A.I.; Shuja, J.; Ahmad, Z.; Dhaou, I.B.; Alsharekh, M.F. LiMPO: Lightweight mobility prediction and offloading framework using machine learning for mobile edge computing. *Clust. Comput.* **2023**, *26*, 99–117. [CrossRef]
29. Xu, X.L.; Fang, Z.J.; Qi, L.; Dou, W.; He, Q.; Duan, Y. A deep reinforcement learning-based distributed service off loading method for edge computing empowered internet of vehicles. *Chin. J. Comput.* **2021**, *44*, 2382–2405.
30. Zhang, X.; Wu, W.; Wang, J.; Liu, S. BiLSTM-based Federated Learning Computation Offloading and Resource Allocation Algorithm in MEC. *ACM Trans. Sens. Netw.* **2023**, *19*, 1–20. [CrossRef]
31. Meng, L.; Wang, Y.; Wang, H.; Tong, X.; Sun, Z.; Cai, Z. Task offloading optimization mechanism based on deep neural network in edge-cloud environment. *J. Cloud Comput.* **2023**, *12*, 76. [CrossRef]
32. Ai, Z.; Zhang, W.; Li, M.; Li, P.; Shi, L. A smart collaborative framework for dynamic multi-task offloading in IIoT-MEC networks. *Peer—Peer Netw. Appl.* **2023**, *16*, 749–764. [CrossRef]
33. Zhang, J.; Wang, J.; Yuan, Z.; Zhang, W.; Liu, L. Offloading demand prediction-driven latency-aware resource reservation in edge networks. *IEEE Internet Things J.* **2023**, *10*, 13826–13836. [CrossRef]
34. Yao, W. The application of artificial intelligence in the internet of things. In Proceedings of the 2019 International Conference on Information Technology and Computer Application (ITCA), Guangzhou, China, 20–22 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 141–144.

35. Yan, J.; Bi, S.; Zhang, Y.J.A. Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 5404–5419. [CrossRef]
36. Miao, Y.; Wu, G.; Li, M.; Ghoneim, A.; Al-Rakhami, M.; Hossain, M.S. Intelligent task prediction and computation offloading based on mobile-edge cloud computing. *Future Gener. Comput. Syst.* **2020**, *102*, 925–931. [CrossRef]
37. Qin, W.; Chen, H.; Wang, L.; Xia, Y.; Nascita, A.; Pescapè, A. MCOTM: Mobility-aware computation offloading and task migration for edge computing in industrial IoT. *Future Gener. Comput. Syst.* **2024**, *151*, 232–241. [CrossRef]
38. Shewalkar, A.; Nyavanandi, D.; Ludwig, S.A. Ludwig S A. Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. *J. Artif. Intell. Soft Comput. Res.* **2019**, *9*, 235–245. [CrossRef]
39. Zhou, X.; Bai, G.; Tao, J.; Xu, B. An improved method to search all minimal paths in networks. *IEEE Trans. Reliab.* **2023**, *72*, 1420–1431. [CrossRef]
40. Zhou, H.; Wang, Z.; Min, G.; Zhang, H. UAV-aided computation offloading in mobile-edge computing networks: A Stackelberg game approach. *IEEE Internet Things J.* **2022**, *10*, 6622–6633. [CrossRef]
41. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017. Advances in neural information processing systems.
42. Li, S.; Wu, Y.; Cui, X.; Dong, H.; Fang, F.; Russell, S. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In Proceedings of the AAAI conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; AAAI: Palo Alto, CA, USA, 2019; pp. 4213–4220.
43. Ra, M.R.; Sheth, A.; Mummert, L.; Pillai, P.; Wetherall, D.; Govindan, R. Odessa: Enabling interactive perception applications on mobile devices. In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, Bethesda MD, USA, 28 June–11 July 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 43–56.
44. Wang, J.; Hu, J.; Min, G.; Zhan, W.; Zomaya, A.Y.; Georgalas, N. Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Trans. Comput.* **2021**, *71*, 2449–2461. [CrossRef]
45. Zhang, T.; Xu, Y.; Loo, J.; Yang, D.; Xiao, L. Joint computation and communication design for UAV-assisted mobile edge computing in IoT. *IEEE Trans. Ind. Inform.* **2019**, *16*, 5505–5516. [CrossRef]