

## Article

# Edge Collaborative Online Task Offloading Method Based on Reinforcement Learning

Ming Sun <sup>1,2,\*</sup>, Tie Bao <sup>1</sup>, Dan Xie <sup>1</sup>, Hengyi Lv <sup>2</sup>  and Guoliang Si <sup>2</sup>

<sup>1</sup> College of Computer Science and Technology, Jilin University, Changchun 130012, China; baotie@jlu.edu.cn (T.B.); xiedan@jlu.edu.cn (D.X.)

<sup>2</sup> Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, Changchun 130033, China; lv\_hengyi@163.com (H.L.); siguol@163.com (G.S.)

\* Correspondence: sunm19@mails.jlu.edu.cn

**Abstract:** With the vigorous development of industries such as self-driving, edge intelligence, and the industrial Internet of Things (IoT), the amount and type of data generated are unprecedentedly large, and users' demand for high-quality services continues to increase. Edge computing has emerged as a new paradigm, providing storage, computing, and networking resources between traditional cloud data centers and end devices with solid timeliness. Therefore, the resource allocation problem in the online task offloading process is the main area of research. It is aimed at the task offloading problem of delay-sensitive customers under capacity constraints in the online task scenario. In this paper, a new edge collaborative online task offloading management algorithm based on the deep reinforcement learning method OTO-DRL is designed. Based on that, a large number of simulations are carried out on synthetic and real data sets, taking obstacle recognition and detection in unmanned driving as a specific task and experiment. Compared with other advanced methods, OTO-DRL can well realize the increase in the number of tasks requested by mobile terminal users in the field of edge collaboration while guaranteeing the service quality of task requests with higher priority.

**Keywords:** self-driving; edge synergy; reinforcement learning



**Citation:** Sun, M.; Bao, T.; Xie, D.; Lv, H.; Si, G. Edge Collaborative Online Task Offloading Method Based on Reinforcement Learning. *Electronics* **2023**, *12*, 3741. <https://doi.org/10.3390/electronics12183741>

Received: 22 July 2023

Revised: 30 August 2023

Accepted: 1 September 2023

Published: 5 September 2023



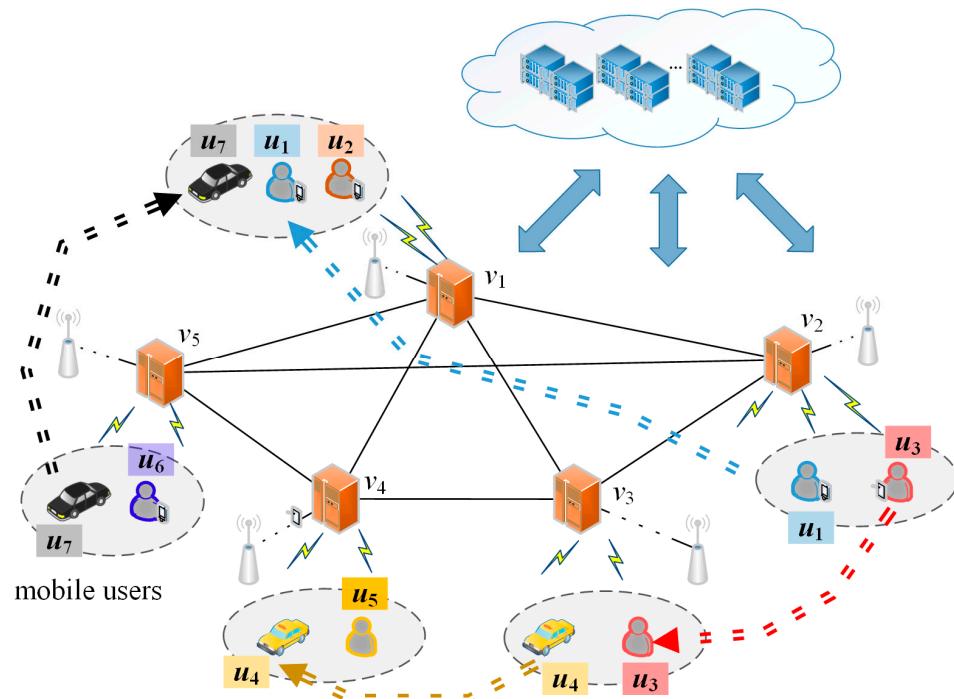
**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

With the unprecedented amount and variety of data generated, users' demand for high-quality services continues to increase. Edge computing is an emerging paradigm that provides storage, computing, and network resources between traditional cloud data centers and end devices. In edge computing, the basic infrastructure is edge nodes, including industrial switches, controllers, routers, video surveillance cameras, and embedded servers [1–5]. Because IoT devices are constantly generating data, analytics must be highly time-sensitive. An important issue is to find a provisioning strategy for edge nodes that can reduce the monetary cost of edge resources and reduce transmission delays for users. Researchers have extensively explored resource provisioning for user workloads in most existing studies, such as offloading computing tasks directly to individual edge nodes or the cloud. This paper considers the task offloading problem of delay-sensitive users under capacity constraints in the online task scenario. It focuses on the resource allocation problem in the online task offloading process based on the edge node collaboration method. The resource allocation problem is determining the allocation of resources on the edge node to multiple users under user deadlines and edge node computing resources to minimize the total cost. Collaboration means an edge node can use rented edge nodes to provide services jointly. Our goal is to improve the cost efficiency of edge computing for network operators while maintaining the quality of service for users.

We take the actual scenario as an example to explain the motivation of the research problem in detail. Some of the definitions and symbolic representation methods involved need to be clearly stated and will be explained in the follow-up problem description section.

We specify the example with an arbitrary number  $n$  of edge nodes and  $m$  of users, and we take  $n = 5$  and  $m = 7$  as the example. We assume there are five heterogeneous edge nodes ( $v_{(1-5)}$ ), and each edge node provides different computing resources to users. At the same time, the computing capabilities of edge nodes are limited, and edge nodes from different suppliers support collaborative services. We assume that within a period, the users passing through this area are ( $u_{(1-7)}$ ), and for each user, its connection range is within a specific area, as shown by the gray circle in Figure 1. In this area, users can offload tasks to corresponding edge nodes. We assume that the connection and location of the edge node have been fixed by a third-party service provider or cloud data center, and the edge node receives and responds to the offload request of the terminal device within the service range and returns the result to the user after processing on the server. We assume that users can offload tasks to their nearest edge nodes, and when the tasks are offloaded to the same service entity, the running time on the service entity will grow linearly with the increase in the number of loads.



**Figure 1.** Example of motivation for online edge-cloud collaborative task offloading. (The grey area represents the coverage range of the edge server; the dashed lines represent the movement trajectories of users).

According to the scenario above, the dynamic changes in the network caused by mobile edge devices have a significant impact on the user's choice of offloading location and the resulting performance. We use the following example to illustrate this effect. As shown in Figure 1, user  $u_4$ , the user's trajectory is from the area where edge node  $v_3$  is located to the area where  $v_4$  is located. We assume that node  $v_4$  has better computing power than node  $v_3$ . For user  $u_4$ , an offloading scheme is to offload the workload of user  $u_4$  to the edge node  $v_3$  at the new location. In this scheme, the cost of user  $u_4$  is the sum of the task processing of node  $v_3$  and the transmission cost of the resulting feedback after completion. However, due to the dynamic movement of users on the edge network, assuming that during the task unloading process of user  $u_4$ , user  $u_3$  reaches area  $v_3$  at the same time and offloads the task to the edge node  $v_3$ . If user  $u_3$  has a higher task priority than  $u_4$  and both users attempt to offload tasks simultaneously, it could lead to a sudden increase in processing costs and delay for  $u_4$ 's task. Another solution is to offload the workload to the edge node  $v_4$  at the target position of the mobile trajectory. This solution can effectively

reduce the delay of task processing, but the resulting communication overhead cannot be ignored. Therefore, for task offloading under online mobile users, the user's offloading location must be considered, and the network dynamic changes caused by device mobility must be considered comprehensively.

In this paper, we research the problem of online task offloading in the mobile scenario of multiple users, which is an original proposal. We jointly optimize user request task delay and energy consumption, and we consider a more realistic and complex scenario on this basis, that is, multi-user mobility.

Our problem poses the following unique challenges: (1) Due to the limited and heterogeneous computing power of edge nodes, when several groups of users arrive with workloads of different sizes, finding a feasible task offloading strategy can be accomplished. It is a challenging task to complete the workload for the user within the deadline. (2) In this paper's definition of the problem scenario, end users are mobile, and their action trajectories are random and time-varying. During the movement process, edge nodes can connect with some users, and users must offload their workload to the edge located in their effective area. It takes work to quickly feedback the processing results of requests to mobile users simultaneously. (3) The collaborative work of edge nodes can reduce the delay for users, but the communication cost may increase. It is not trivial to achieve efficient edge resource allocation while satisfying multiple mobile user requests with minimal cost to balance the trade-off between cost and latency.

This paper focuses on the online task offloading problem in the edge collaboration scenario and realizes the joint optimization of mobile user delay and energy consumption under capacity and computing power constraints. The main contributions of this paper are as follows:

- (1) This paper discusses the problem of online task offloading for multiple mobile users and mainly studies the scenario of edge nodes working together in mobile edge computing. On this basis, a reward evaluation method based on deep reinforcement learning is proposed to realize the online tasks of multiple mobile users. Task offloading jointly optimizes overall latency and energy consumption under the constraints of edge physical resources.
- (2) According to the characteristics of user mobility, a trajectory prediction method based on multi-user movement is designed to reduce the delay caused by user mobility. On this basis, centering on the task request user, a task optimization scheduling mechanism based on task multi-dimensional characteristics and marginal resource conditions is proposed, which avoids invalid states in the decision-making module and approximates the policy in the decision-making module according to the introduced feasibility mechanism.
- (3) Finally, we conduct experiments based on synthetic and natural simulation datasets. We compare the proposed joint optimization method with several state-of-the-art methods in the synthetic simulation dataset section. We also evaluate the experimental results from different perspectives to provide corresponding conclusions. On this basis, we take obstacle detection and recognition in unmanned driving as a specific task and further verify the effectiveness of the proposed method based on real data sets. The experimental results show that the scheme can well realize the increase in the number of tasks requested by mobile terminal users in the edge collaborative service area and simultaneously guarantee the service quality of task requests with higher priority.

## 2. Related Work

In order to effectively coordinate computing resources in the end-edge-cloud collaborative computing paradigm and free mobile devices from limited computing power and energy supply, both the industry and academia regard computing offloading as a promising solution. Computing offloading is one of the hot research issues in the field of mobile edge computing. Many researchers have also performed much exploration on

this issue. In the existing research, mobile edge computing application algorithms with different theoretical characteristics have been proposed. In computing offload for mobile edge computing, we must address two main related problems. The first is deciding when to offload computing tasks from devices to servers for processing; the second is how to allocate server resources to meet user needs reasonably. In order to solve the above problems, some existing works have proposed models and algorithms based on different optimization objectives. Mukherjee et al. [6] considered the transmission delay and service rate from fog to cloud, and jointly optimized the number of tasks offloaded to adjacent fog nodes and the allocation of communication resources offloaded to remote cloud through semi-definite relaxation. Sarkar et al. [7] used the task classification strategy to propose a dynamic task allocation strategy to allocate the required tasks, minimize the delay, and meet the deadline. Yang et al. [8] proposed a feed-forward neural network model based on multi-task learning to solve binary offloading decision-making and computing resource allocation problems effectively. This method transforms the original problem into a mixed-integer nonlinear programming problem and uses the MTFNN model trained offline to derive the optimal solution. Compared with traditional optimization algorithms, this method has a lower computational cost and is significantly better than traditional methods in terms of computational time and inference accuracy. Zhan et al. [9–11] considered the mobility in the process of task offloading in the computational offloading problem, converted the original global optimization problem into multiple local optimization problems, and proposed a heuristic mobile perceptual offloading algorithm used to obtain an approximate optimal offloading scheme. Zhou et al. [12] proposed a reliability stochastic optimization model based on dynamic programming to deal with the dynamic and stochastic characteristics of the vehicle network and ensure the reliability of vehicle computing offloading. At the same time, they also proposed an optimal data transmission scheduling mechanism that considers the randomness of vehicular infrastructure communication and can maximize the lower bound.

In addition to the above work, in consideration of energy consumption, there are various solutions to the task offloading problem in different environments and scenarios. Under delay constraints, Zhang et al. [13] designed an energy offloading strategy using the artificial fish swarm algorithm, which considers link conditions and effectively reduces device energy consumption. However, the algorithm complexity is high. In a multi-resource environment, Xu et al. [14] proposed an energy-minimized particle swarm task scheduling algorithm to match multiple resources and reduce the energy consumption of edge terminal devices. Wei et al. [15–17] divided the task offloading problem into mobile management problems and energy saving problems and used a greedy algorithm to minimize the energy consumption of mobile devices. Lu et al. [18] provided an efficient resource allocation scheme to minimize the total cost of multiple mobile users by considering three different cases. Yu et al. [19] studied the problem of task offloading in ultra-dense network scenarios. They proposed a task offloading algorithm based on Lyapunov optimization theory that effectively reduces the total energy consumption of base stations. Aiming at the problems of high energy consumption and computing power that mobile social platforms may cause, Guo et al. [20] proposed an energy consumption optimization model based on the Markov decision process, which considers the network status of different environments and dynamically selects the best network. Access and refresh downloads in the best image format to reduce power consumption. In order to solve the privacy leakage problem that may occur in offloading decisions, Liu et al. [21] studied the offloading problem based on deep learning. They proposed a deep learning-based offloading algorithm group sparse beamforming framework to optimize network power consumption. These task offloading decisions have achieved the purpose of reducing the delay time. However, they do not consider the impact of the energy consumption of the terminal device during the task offloading calculation process, and the terminal device may not be able to operate normally due to insufficient power.

Some researchers jointly proposed related solutions to the problem of joint optimization of delay and energy consumption. Gao et al. [22] proposed a joint computing offloading and priority task scheduling scheme in mobile edge computing that uses a dynamic priority level task scheduling algorithm while considering the urgency of the task and the idleness of the edge server. This can reduce the task completion time and improve the service quality by assigning the task to the edge server. Kim et al. [23] established the problem as a linear integer optimization problem to optimize delay and resource costs. This paper introduces a system, MoDEMS, that optimizes the deployment of edge computing based on user mobility, and proposes a Seq-Greedy heuristic algorithm to generate a migration plan that minimizes system cost and user delay. Ale et al. [24] proposed an end-to-end deep reinforcement learning method to offload services to the best edge server and allocate the optimal computing resources to maximize the completion of tasks before their respective deadlines and minimize energy consumption. Hazra et al. [25] proposed a heuristic-based transmission scheduling strategy that transmits according to the importance of the generated task. A graph-based task offloading strategy is introduced, which uses constrained mixed linear programming to deal with high traffic in peak-period scenarios while maintaining energy and delay constraints. Zhang et al. [26] considered the effect of task priority on task offloading when solving the offloading decision problem. In order to better meet user needs, they proposed a method based on the importance model, which considered the differences between different user tasks. They combined the maximum constraint delay of completing the task and the size of computing resources required to complete the task. As the two main factors of task importance, the characteristics of the task are considered comprehensively. Yu et al. [27–29] integrated mobility prediction in offload strategy and resource allocation methods, combining offload strategy and mobility management modules. This method can intelligently allocate tasks according to the user's mobile mode to allocate tasks to locations with better network conditions in the future as much as possible to reduce energy consumption. From the above work, the individual differences of the task determine its importance, which is determined by the size of the required computing resources and the maximum tolerant delay. At the same time, user mobility is closely related to service quality. Frequent movement may cause the terminal device to leave the service range of the server, thereby interrupting the service. In addition, when offloading tasks are forwarded between different servers, a considerable transmission delay will be generated, resulting in an untimely service response and poor effect. Therefore, in this section, when solving the offloading problem of online task computing, multi-user mobility and task characteristic differences are considered comprehensively.

### 3. Problem Description

In the context of mobile edge computing, the third section delves into the comprehensive exploration of the system's intricate mechanisms. This section, divided into three subsections, expounds on the nuances of the system model, transfer model, and computational model.

#### 3.1. System Model

In this subsection, the mobile edge computing system adopts the three-layer architecture of cloud, edge, and terminal. We assume that the set of edge nodes in a specific activity area is  $M = \{m_j\}$ . These edge nodes are connected to a base station with limited computing power and storage capacity. We use the set  $U = \{u_i\}$  to denote mobile users served by edge nodes. The mobility of end users is described by the set  $\{u_i^r, u_i^d\}$ ,  $u_i^r$  represents the mobile rate of the end user;  $u_i^d$  represents the direction of movement of the end user. In order to better capture the movement of each user, the system is assumed to operate in period slots, which are discretized into a time series  $t \in T = \{0, 1, 2, \dots, T\}$  [15,30]. The trajectories of the terminal devices will be dynamically updated at the beginning of each time slot. Here we use  $V$  to represent the task request set,  $V = \{v_i\}$ , where  $|v_i|$  represents the size of task  $v_i$ , that is, the number of bits contained. We assume that at the beginning of

each time slot  $t \in T$ , an end user  $u_i \in U$  will generate an indivisible task  $v_i$ , and each end user with mobility sends a task request to the edge node. Here we use  $x_{ij}(t) = 1$  to indicate that user  $u_i$  offloads task  $v_i$  to edge server  $m_j$  at time  $t$ . Otherwise,  $x_{ij}(t) = 0$ . For each edge node, use  $V_{m_j}$  to represent the task request set placed on the edge server  $m_j$ , where  $V_{m_j} = \{v_i | m_j \leftarrow v_i\}$ .

### 3.2. Transfer Model

This section presents a transfer model for user task offloading to edge nodes. Here, we define the transmission rate of the offload link as  $r_{u_i, m_j}$ , and the transmission power as  $p(u_i, m_j)$ . The specific calculation method is shown in Formula (1).

$$r_{u_i, m_k} = W \times \log_2 \left( 1 + \frac{|h_{m_s}|^2}{\sigma^2} \times p(u_i, m_j) \right) \quad (1)$$

Here we use  $|h_{m_s}|^2$  to denote the channel gain between users  $u_i$  and  $m_j$ ,  $\sigma^2$  to denote the noise power, and let  $p(u_i^k, m_k)$  denote the transmission from  $u_i$  to device  $m_j$  power.

### 3.3. Computational Model

This section defines computing models for end users and edge servers, respectively. For end users, define two local computing and task offloading queues, respectively. For the starting moment of the  $t$  time slot, the length of the computing queue is represented by  $l_{u_i}^c(t)$ , and the unloading queue is represented by  $l_{u_i}^t(t)$ , indicating that the queue length is represented by  $|l_{u_i}^t(t)|$ , and this paper reflects the load condition through the length of the queue. For each end user, we define its processor parameters as follows:  $f_{u_i}^c$  represents the CPU frequency of the end user  $u_i \in U$  processor, that is, the computing power of the terminal device  $u_i$ , and  $p_{u_i}^w$  and  $p_{u_i}^c$  represent the standby power and computing power of the device's  $u_i$  processor. The standby power is used to calculate the standby capacity loss in the process of waiting for the unloading result, and the calculation power is used to calculate the energy consumption required to process task  $v_i$ , and the two are used for the total benefit evaluation. For the server side, we assume that each edge server has a task queue for processing tasks offloaded to the server, and here a first-in-first-out scheduling method is adopted.  $l_{m_j}^v(t)$  indicates the queue at the beginning of the  $t$  th time slot of the task queue, where the queue length is represented by  $|l_{m_j}^v(t)|$ , which is used to reflect the server's load. At the beginning of each time slot, the server node will broadcast the load condition of the task queue to all terminal devices in the service area for the terminal devices to make unloading decisions. Here we use  $f_{m_j}^c$  to represent the computing power of the server node  $m_j \in M$ , that is, the CPU frequency of the edge service node processor.

Here, we define the edge servers in the area that can offload services for user systems as  $S$ , and  $m_j \in M$  is satisfied for  $\forall m_j \in S$ . We use  $r_{m_j}$  to represent the service radius of edge server  $m_j$ . We evaluate the process through the delayed income for the user's task offloading delay. The terminal device  $u_i$  must first determine that there are feasible nodes in the server set that can provide offloading services under the current location. The task cannot be offloaded to the edge side if there is no feasible node. The process of constructing the feasible edge node set  $S$  is as follows: (1) Select  $m_j \in M$ , and calculate the distance  $d(p_{u_i}, p_{m_j})$  between the end user  $u_i$  and the edge server  $m_j$  based on their current locations. (2) Compare the service radius  $r_{m_j}$  of the edge server with the distance  $d(p_{u_i}, p_{m_j})$ . If  $d(p_{u_i}, p_{m_j}) \leq r_{m_j}$ , then add  $m_j$  into the feasible set  $S$ . (3) Repeat the above steps until all edge nodes in the set  $M$  are traversed. Based on the feasible edge node set  $S$  constructed above, a delay calculation is performed on all edge nodes in  $S$ , respectively. The terminal device  $u_i$  generates a task  $v_i(t)$  at the beginning of the  $t \in T$  time slot, and the task size is  $|v_i(t)|$ .

### 3.3.1. Execute on the Local Device

We define  $D_{u_i}^l$  as the local execution delay of task  $v_i$  of end-user  $u_i$ . The specific calculation method is shown in Formula (2):

$$D_{u_i}^l(t) = \sum_{v_i \in V_{m_j}}^{|l_{u_i}^c(t)|} |v_i(t)| / f_{u_i}^c \quad (2)$$

The length of the  $t$  slot calculation queue is  $l_{m_j}^t(t)$ , and the processing capability of the mobile terminal equipment is  $f_{u_i}^c$ .

We define  $E_{u_i}^l(t)$  as the energy consumption of tasks executed locally, and the specific calculation method is shown in Formula (3):

$$E_{u_i}^l(t) = f_{u_i}^c \times D_{u_i}^l(t) \quad (3)$$

Among them,  $D_{u_i}^l(t)$  is the local processing delay calculated by the Formula (2), and  $f_{u_i}^c$  indicates the computing capability of the mobile terminal device  $u_i$ .

### 3.3.2. Offload Processing Delay

We define  $d_{u_i}^e(t)$  to represent the execution delay of the task on the server, where the length of the unloading queue of the edge server  $m_j$  is  $l_{m_j}^t(t)$ , and the specific calculation is as shown in Formula (4):

$$d_{u_i}^e(t) = \sum_{v_i \in V_{m_j}}^{|l_{m_j}^t(t)|} |v_i(t)| / f_{m_j}^c \quad (4)$$

On this basis, we define the delay of task transmission,

$$d_{u_i}^t(t) = \sum_{v_i \in V_{m_j}}^{|l_{m_j}^t(t)|} |v_i(t)| / r_{u_i, m_k} \quad (5)$$

Here, we define  $D_{u_i}^e(t)$  to represent the total delay of task unloading. Usually, the magnitude of returned result data after task processing is small, so this paper does not consider the calculation time delay, and the transmission time of the returned result is delayed. Therefore, we have the following:

$$D_{u_i}^e(t) = d_{u_i}^e(t) + d_{u_i}^t(t) \quad (6)$$

In order to facilitate the location decision of the end user during the unloading process, this section measures it by defining the delayed benefit, here we use  $D_\Delta$  to express it, and the specific calculation method is shown in Formula (7):

$$D_\Delta(t) = D_{u_i}^l(t) - D_{u_i}^e(t) \quad (7)$$

We define  $e_{u_i}^e(t)$  as the energy consumption of tasks offloaded to the server, where  $\sigma_{u_i}$  represents the standby energy consumption of the terminal  $u_i$  waiting for the data processing results, and  $d_{u_i}^e(t)$  represents the execution delay on the server, so the calculation method of  $e_{u_i}^e(t)$  is shown in Formula (8):

$$e_{u_i}^e(t) = \sigma_{u_i} \times d_{u_i}^e(t) \quad (8)$$

We define  $e_{u_i}^t(t)$  as the energy consumption of tasks transmitted to the offload server, and the specific calculation method is shown in Formula (9):

$$e_{u_i}^t(t) = p(u_i^k, m_k) \times d_{u_i}^t(t) \quad (9)$$

Therefore, the total energy consumption of tasks offloaded to the edge server is  $E_{u_i}^e(t)$ , and the specific calculation formula is shown in (10):

$$E_{u_i}^e(t) = e_{u_i}^e(t) + e_{u_i}^t(t) \quad (10)$$

Here, we define the energy consumption benefit from measuring it, and we express it as  $E_\Delta$ , and the specific calculation method is shown in Formula (11):

$$E_\Delta(t) = E_{u_i}^l(t) - E_{u_i}^e(t) \quad (11)$$

In order to balance the relationship between delay and energy consumption, we define the variable  $\varepsilon \in [0, 1]$  to represent the preference factor, which determines whether the optimization goal is more inclined to reduce delay or reduce energy consumption.  $\Psi(t)$  represents the total benefit evaluation result. The specific calculation method is shown in Formula (12).

$$\Psi(t) = \varepsilon \times D_\Delta(t) + (1 - \varepsilon) \times E_\Delta(t) \quad (12)$$

In this paper, we mainly study the problem of online task offloading in the edge collaborative operation scenario. With the optimization goal of maximizing revenue under limited resources, a new online task offloading method based on deep reinforcement learning is proposed. Optimizing total revenue is subject to cost constraints.

$$\text{maximize}_{t=0}^T \sum_{j=1}^{|M|} \sum_{i=1}^{|U|} \Psi(t) \quad (13)$$

$$\text{s.t. } |l_{u_i}^c(t)| \leq \tau, |l_{m_j}^t(t)| \leq \Gamma, \forall m_j \in M \quad (14)$$

$$x_{ij}(t) \in \{0, 1\}, \forall u_i \in U, \forall m_j \in M \quad (15)$$

Among them, the Formula (13) is the optimization objective, and the Formulas (14) and (15) are the constraints. Formula (14) is a physical resource constraint, which means that the computing resource queue provided by the terminal device cannot exceed the threshold  $\tau$ , and, at the same time, the computing resource queue provided by the edge server cannot exceed the threshold  $\Gamma$ . Formula (15) shows the user  $u_i$  whether to use service  $m_j$  at time slot  $t$ .

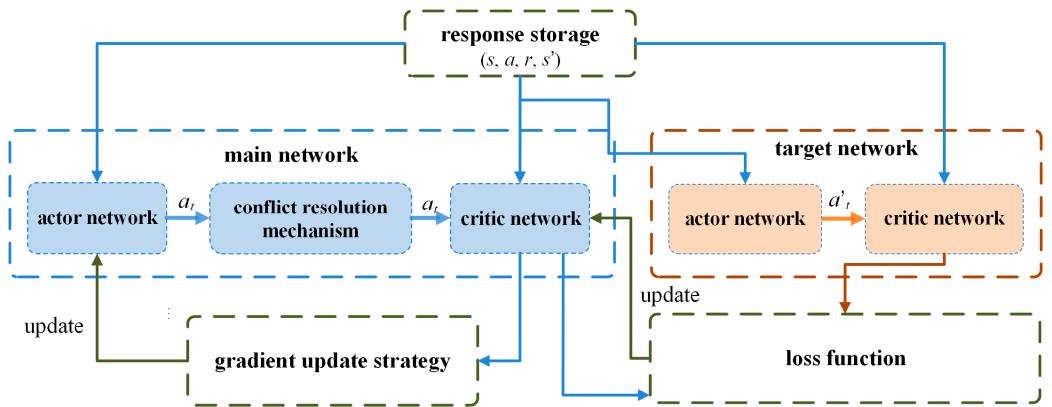
#### 4. Edge Collaborative Online Task Offloading Strategy Based on Deep Reinforcement Learning

In order to minimize the total delay of the current user set over a continuous period, a novel decentralized dynamic service facility management framework based on deep reinforcement learning is designed in this section to achieve lower latency under physical resource and cost constraints.

##### 4.1. Overall Policy Framework

In this subsection, a novel edge online task offloading based on deep reinforcement learning (OTO-DRL) management framework based on deep reinforcement learning is designed to achieve higher overall profit under physical resource and cost constraints. Figure 2 shows the overall structure of the OTO-DRL framework.

Since the decision-making process in online task offloading is a stochastic optimization process, this section studies the framework based on the deep deterministic policy gradient (DDPG) algorithm. In order to concisely and accurately describe the current environment and state space, we need to consider the task workload distribution on the edge servers and the states they provide to users. Therefore, we design the state and action spaces, reward functions, and state transition strategies in the reinforcement learning framework. The definition of the reinforcement learning design is shown below.



**Figure 2.** The overall structure of the OTO-DRL framework.

**Definition 1** (State Space). *The state space describes the current environmental state of the mobile edge network, and it is a vector defined as  $s_t = [\hat{v}_t, \mathbf{p}_t, \hat{r}_t]$ .  $\hat{v}_t = (v_1(t), v_2(t), \dots, v_i(t), \dots)$  represents the task offloading sizes the terminal devices send to the server node in the  $t$ -th time slot. For each terminal device, when  $u_i$  does not send an offloading request to an edge server at the beginning of the  $t$ -th time slot, the value of  $v_i(t) = 0$ . In that case, the set  $\mathbf{p}_t = \{p_1(t), p_2(t), \dots, p_i(t)\}$  corresponding to the revenue of  $v_i(t)$  in  $\hat{v}_t$  is the priority set of the offloading tasks that the terminal device sends to the edge server at the beginning of the  $t$ -th time slot. If the terminal device  $u_i$  does not send an offloading request to a server node  $m_j \in M$  at the beginning of the  $t$ -th time slot, the value of the element at the corresponding position of  $u_i$  in the set  $p_i(t)$  is 0; otherwise, the value is equal to the priority of the offloading task. In the state space  $s_t$ , the remaining computing resources of the edge server at the beginning of the  $t$ -th time slot are represented by  $\hat{r}_t$ .*

**Definition 2** (Action Space). *The action space describes the behavioral decisions of the agent, denoted as  $a_t = [\hat{\mathbf{m}}_1, \hat{\mathbf{m}}_2, \dots, \hat{\mathbf{m}}_h, \dots, \hat{\mathbf{m}}_n]_t$ , which represents the migration strategy of tasks in time slot  $t$ .  $\hat{\mathbf{m}}_h = [\hat{m}_h(t)^-, \hat{m}_h(t)^+]$  represents the range of selection of edge servers during the migration process of task  $v_h$  in time slot  $t$ . Here,  $\hat{m}_h(t)^- \in \{0, 1\}$ , where 0 indicates local execution, and 1 indicates offloading to the currently connected edge server. For each service, the optional edge servers are represented by a range of consecutive edge server numbers  $[\hat{m}_h(t)^-, \hat{m}_h(t)^+]$ , where  $\hat{m}_h(t)^-$  represents the minimum number of edge servers that can be selected during task migration, and  $\hat{m}_h(t)^+$  represents the maximum number of collaborating edge servers.*

Since the problem we study here is an online learning process, the value of the reward cannot directly determine the final total profit of multiple mobile users in each time slot. Taking the  $t$ -th time slot as an example, the reward for completing a task considers current and future states. We define the essential reward value as  $\varepsilon_{v_i}$ , and the reward for completing a single task is the product of the fundamental reward value and the task priority. Taking task  $v_i$  generated by terminal device  $u_i$  at the beginning of the  $t$ -th time slot as an example, the reward value that can be obtained by completing  $v_i$  is as follows:

$$R_{v_i}(t) = \varepsilon_{v_i} \times p_i(t) \quad (16)$$

For an edge server  $m_j$  during a time slot, we define the variable  $\mathbf{R}_{m_j}(t)$  as the instantaneous profit, which is the total reward the edge server can obtain after completing multiple tasks. We define the set  $l_{m_j}^v(t)$  as the task queue of server node  $m_j$  at the beginning of the  $t$ -th time slot. Assuming that server node  $m_j$  completes tasks with indices 1 to  $n$  in the task queue during the  $t$ -th time slot. The formula for calculating the instantaneous profit is as follows:

$$\mathbf{R}_{m_j}(t) = \sum_{i=1}^n l_{m_j}^{v_i}(t) p_i(t) \times \varepsilon_{v_i} \quad (17)$$

We define the variable  $\hat{\mathbf{R}}_{m_j}(t)$  as the expected future profit, which is the reward that can be obtained from the offloading requests that arrive at server node  $m_j$  at the beginning of time slot  $t$ , in the future. Here, we define  $\omega_{v_i} \in \{0, 1\}$  as whether to process task  $v_i$ , and the specific calculation formula is as follows:

$$\hat{\mathbf{R}}_{m_j}(t) = \sum_{i=1}^{l_{m_j}^v(t)} R_{v_i}(t) \times \omega_{v_i}(t) \quad (18)$$

We define the variable  $\bar{\mathbf{R}}_{m_j}(t)$  as the expected future loss, which is the total reward value of rejecting offloading tasks after the offloading scheduling at server node  $m_j$  at the beginning of time slot  $t$ . The calculation formula is as follows:

$$\bar{\mathbf{R}}_{m_j}(t) = \sum_{i=1}^{l_{m_j}^v(t)} R_{v_i}(t) \times (1 - \omega_{v_i}(t)) \quad (19)$$

**Definition 3 (Reward).** *The reward value is determined by the above multiple variables, considering the current profit of the edge server, as well as the expected future profit and loss after performing offloading scheduling operations. The reward is defined as shown in (20).*

$$\mathbf{R}(t) = \mathbf{R}_{m_j}(t) + \hat{\mathbf{R}}_{m_j}(t) + \bar{\mathbf{R}}_{m_j}(t) \quad (20)$$

#### 4.2. Load Balancing-Based Multi-Task Offloading Conflict Resolution Mechanism

This paper aims to minimize the total profit while enabling multiple mobile users to perform online task offloading. For each user's task offloading request, the decision depends on observing the mobile edge network environment from their respective perspectives during each training process. However, the mobile edge computing system has no prior knowledge, meaning each service needs to know the size of the user's data or trajectory. At the same time, the entire process is online and model-free, and multiple users move irregularly and independently during the learning process. Therefore, when the trajectories of multiple users are similar or overlap in the learning process, resource allocation imbalance is prone to occur in the coverage positions of multiple edge servers, increasing computation delay in some areas and decreasing user service quality. In order to maintain the performance of edge network services, this paper analyzes the closely coupled relationship between the activity trajectories of terminal users and edge resource load balancing. It proposes a multi-task offloading conflict resolution mechanism based on edge network load balancing.

We propose a new multi-task offloading conflict resolution mechanism in the decision-making module to avoid invalid states and approximate policies. Our solution mechanism includes two main stages: one is to find edge servers and service requests with uneven load distribution, and the other is to make collaborative job decisions for high-load edge nodes. Algorithm 1 describes the service placement problem, with the input being action  $a_t$  and time slot  $t$ , and the output being the updated service placement strategy. In this process, the algorithm also enables a conflict resolution mechanism to ensure that the service placement for edge servers can be effectively implemented. Based on the decision  $a_t$  given by reinforcement learning under the current time slot, we first perform pre-offloading according to the task requests under each user's predicted trajectory and then check the status of each edge server. If the number of tasks on an edge server exceeds its queue capacity after pre-offloading, it indicates that the server is congested and may cause task-offloading conflicts; otherwise, it indicates that all services offloaded to that server can be completed. Based on the above analysis, we begin to make offloading decisions. For congested edge servers, we first construct the latest task conflict set  $C_{m_j} = \{v_i^{m_j}\}$  on that server, which consists of all tasks that request to be executed on service  $m_j$  simultaneously.

Then, we select the task  $v_i$  with the maximum profit value in the conflict set  $C_{m_j}$  for processing until the number of tasks reaches the queue threshold and update the set  $\bar{C}_{m_j} = C_{m_j}/v_i$ .

**Definition 4** (Collaboration factor).  $\delta(m_k)$  represents the collaboration factor of the adjacent node  $m_k$  to edge server  $m_j$  at time slot  $t$ , defined as shown in Formula (21):

$$\delta(m_k)(t) = d(p_{m_j}, p_{m_k}) + \mu |l_{m_k}^t(t)| \quad (21)$$

For the tasks that have not been processed in the set, we define the collaboration factor based on the node connectivity in the edge network. The core idea is to allocate tasks based on the distance  $d(p_{m_j}, p_{m_k})$  between adjacent edge nodes and the queue length  $|l_{m_k}^t(t)|$ , where  $\mu$  is a tuning parameter. For the edge nodes determined to be collaborative jobs, the results will be returned to the edge server  $m_j$  after completion and then fed back to the terminal users.

---

**Algorithm 1.** Load Balancing-Based Multi-Task Offloading Task Conflict Resolution Method (TCR)

---

Input: The action  $a_t$  time slot  $t$   
 Output: The updated action  $a'_t$  of tasks offloading decisions under the conflict edge servers;

```

1   for each user  $v_i \in V$  do
2       Pre-offloading according to  $a_t$ ;
3   for each edge server  $m_j \in M$  do
4       Calculate the total number of tasks in  $|l_{m_j}^t(t)|$  pre – offloading to  $m_j$ ;
5       if  $|l_{m_j}^t(t)| > \Gamma_{m_j}$  do
6           Construct conflict set  $C_{m_j} = \{v_i^{m_j}\}$ ;
7           Choose task  $v_i$  with maximum  $R_{v_i}(t)$ ;
8           Update set  $\bar{C}_{m_j} = C_{m_j}/v_i$ ;
9           for each task  $v_k \in \bar{C}_{m_j}$  do
10            Update set  $\bar{M} = M/m_j$ ;
11            Select the edge server with minimum  $\delta(m_k)(t)$  in set  $\bar{M}$ 
12        end for
13        Record current state  $a'_t$  and update  $a_t = a'_t$ ;
14    else
15        Keep the original decisions of action  $a_t$ ;
16    end if
17 end for
```

---

#### 4.3. Online Task Offloading Strategy Based on Deep Reinforcement Learning

This section proposes a dynamic service placement strategy based on deep reinforcement learning. Based on the characteristics of the decision-making process, we investigate a solution based on the deep deterministic policy gradient (DDPG) algorithm. The main idea is to use a deep reinforcement learning agent to perform dynamic service placement for multiple mobile users to minimize the total delay. The specific steps are shown in Algorithm 2.

**Algorithm 2.** Online Task Offloading based on Deep Reinforcement Learning (OTO-DRL)

---

Input: Sets of edge nodes  $M$ , services  $V$ , and users  $U$ ;  
 Output: Dynamic service placement scheme  $X$ ;

- 1 Randomly initialize the actor network  $\mu(s|\theta^\mu)$  and critic network  $Q(s, \alpha|\theta^Q)$  with weight  $\theta^\mu$  and  $\theta^Q$ ;
- 2 Initialize the target networks with weights  $\theta^{\mu'} \leftarrow \theta^\mu$  and  $\theta^{Q'} \leftarrow \theta^Q$ ;
- 3 Initialize replay buffer  $B$ ;
- 4 **for** episodes from 1 to  $k$  **do**
- 5     Initialize environmental parameters for edge servers and users, and generate an initial state  $s_1$ ;
- 6     **for** each time slot  $t$  from 1 to  $T$  **do**
- 7         Select an action  $a_t = \mu(s_t|\theta^\mu) + \delta_t$  to determine the destination of migration by running the current policy network  $\theta^\mu$  and exploration noise  $\delta_t$ ;
- 8         Detect migration conflicts and resolve via Algorithm 1;
- 9         Execute action  $a_t$  of each user agent independently, and observe reward  $r_t$  and new state  $s_{t+1}$  from the environment;
- 10        Store the transition tuple  $(s_t, a_t, r_t, s_{t+1})$  into replay buffer  $B$ ;
- 11        Randomly sample a mini-batch of  $I$  transitions  $\{(s_t, a_t, r_t, s_{t+1})\}$  from replay buffer  $B$ ;
- 12        Update the critic network  $Q(s, \alpha|\theta^Q)$  by minimizing the loss function  $L$  in Equation (16);
- 13        Update the actor network  $\mu(s|\theta^\mu)$  by using the sampled policy gradient  $\nabla\theta^\mu J$  in Equation (17);
- 14        Update the target networks :  $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$ ,  $\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$ ;
- 15     **end for**
- 16 **end for**

---

We use sets of edge nodes, services, and users as input and output of a dynamic service placement policy. We initialize the preliminary parameters of the reinforcement learning agent, including the main network, target network, and replay buffer, and begin training. Each edge server independently determines the placement strategy of services (migration or maintaining the original position) through training. We begin by initializing the environment parameters for the edge servers and users, generating an initial state, and starting the training process for a time slot. For each time slot, we select the action value of the current state by running the current decision network  $\theta^\mu$  and variance  $\delta_t$ , which determines the target migration position of each service as  $a_t = \mu(s_t|\theta^\mu) + \delta_t$ . Since user mobility is unstable and autonomous, we detect and resolve any migration conflicts based on Algorithm 2. We execute the action value  $a_t$  for each user agent and observe the reward value and new state from the environment. We then store the state transition tuple  $(s_t, a_t, r_t, s_{t+1})$  of the relevant information in the buffer. The actor and critic networks will be updated based on the value of the mini-batch. The critic network is updated, taking the state and action space as inputs and outputs of the action decision value [30]. Specifically, the critic network approximates the action-value function  $Q(s, \alpha|\theta^Q)$  by minimizing the loss function shown in Formula (22):

$$L = \frac{1}{I} \sum_{\omega=1}^I (r_\omega + \gamma Q'(s'_\omega, \alpha' | \theta^{Q'}) - Q(s_\omega, \alpha | \theta^Q)) \quad (22)$$

For the actor network, it represents the policy parameterized by  $\theta$ , which uses stochastic gradient ascent to maximize  $\nabla\theta^\mu J$ , as shown in Formula (16):

$$\nabla\theta^\mu J \approx \frac{1}{I} \sum_{\omega=1}^I \nabla_\alpha Q'(s_\omega, \alpha | \theta^Q) | \alpha = \alpha_\omega \nabla\theta^\mu \mu(s_\omega | \theta^\mu) \quad (23)$$

Finally, the target network is updated by  $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$  and  $\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$ .

## 5. Evaluation of Experiments

In this paper, we conducted extensive simulations and experiments to study the online task offloading problem among multiple mobile users. We developed a Python framework, including constructing the edge network and handling requests from multiple mobile users. Based on this, we conducted extensive simulations and experiments with obstacle detection and recognition as the specific task in the unmanned driving scenario to study the online task offloading problem among multiple mobile users. After presenting the dataset and experimental settings, we analyzed and presented the results from different perspectives.

### 5.1. Basic Settings of the Experimental Environment

In this subsection, the experiments were conducted in an area with a range of 500 square meters, with 10 mobile edge servers set up within the area. Each edge server's computing capacity range was set from 10 GHz to 15 GHz. The bandwidth between each pair of edge servers was 0.2 GHz, and the transmission power was set to 0.5 W. To analyze the overall revenue for multiple mobile users, we grouped the users into 20, 30, and 40 to construct synthetic datasets. For each group of users, it was assumed that they would send uninterrupted requests for continuous time periods, and the task request data size was randomly varied within the range of [0.1 GB, 0.5 GB]. The hyperparameters were set as shown in Table 1 during the experimental process.

**Table 1.** Simulation parameter settings.

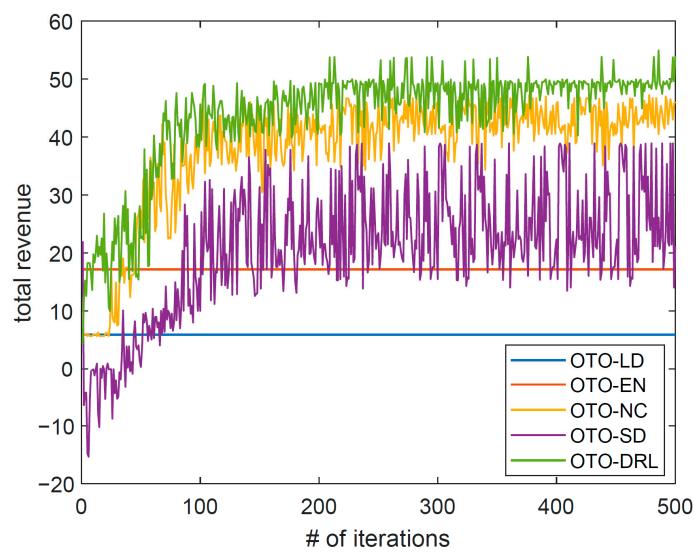
Hyperparameters	Value
Actor network learning rate	0.001
Critic network learning rate	0.002
Reward discount factor $\gamma$	0.9
Soft replacement value $\tau$	0.01
Response storage	200

In addition to the online task offloading algorithm proposed in this section, four baseline algorithms were used as comparison methods, and experiments were conducted on the algorithm under different grouping conditions. The specific comparison algorithms are as follows:

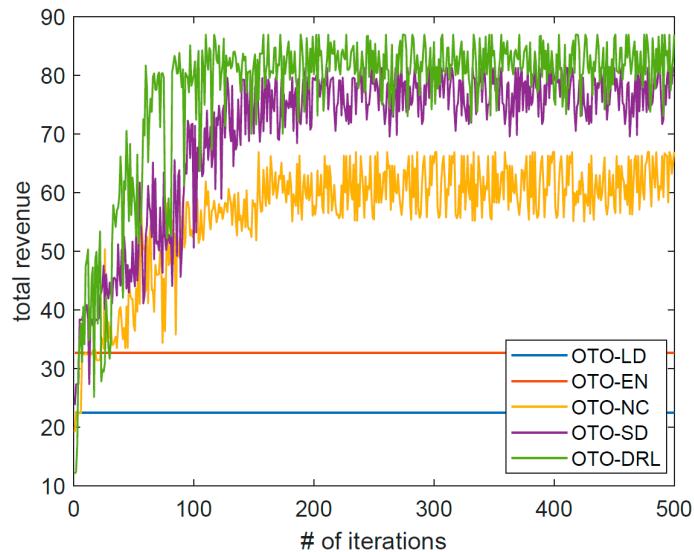
- (1) Online task offloading on local devices (OTO-LD): We iteratively offload user-generated tasks on local devices.
- (2) Online task offloading on edge nodes (OTO-EN): We iteratively offload user-generated tasks on directly connected edge nodes.
- (3) Online task offloading with no cooperation (OTO-NC): We iteratively offload tasks to edge nodes, and in the iterative process, if the edge node is overloaded, the task is returned to be executed locally.
- (4) Online task offloading with shortest distance (OTO-SD): We iteratively offload tasks to edge nodes, and in the iterative process, if the edge node is overloaded, the node closest to the edge node is selected to execute the task.

### 5.2. Convergence Evaluation

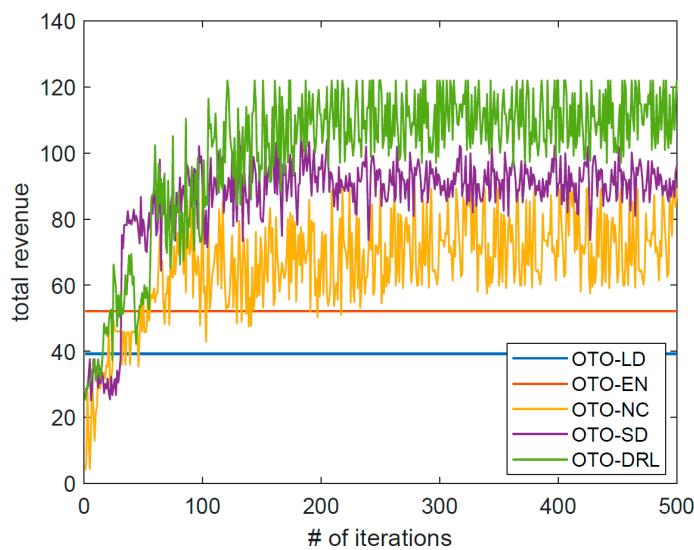
The previous section validated the obstacle detection and recognition algorithm that requires edge computing. In this section, we studied the algorithm's convergence under obstacle detection and recognition for mobile users in three different group sizes. The number of users in each group was 20, 30, and 40, respectively, and each user had 20 trajectories in a period. The results are shown in Figures 3–7. A black dashed line is used to describe the trend of delayed convergence, and it can be seen that the algorithm converges.



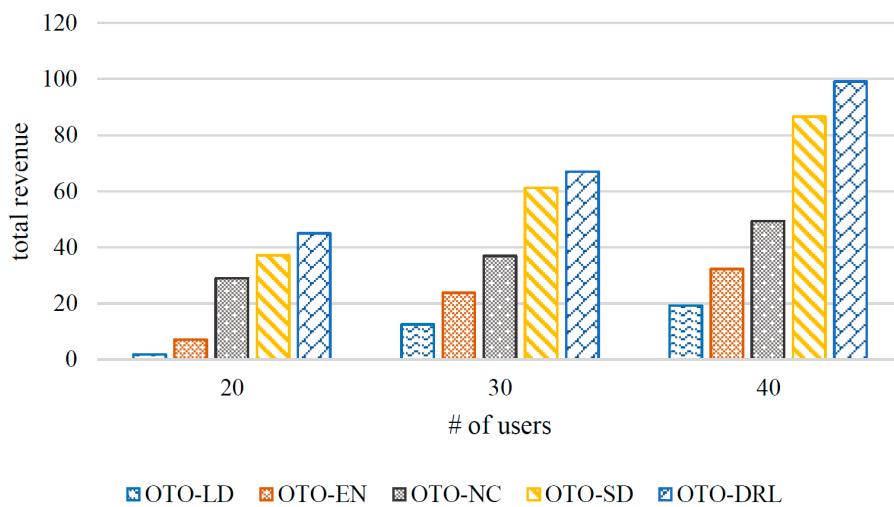
**Figure 3.** Convergence of total revenue for 20 users.



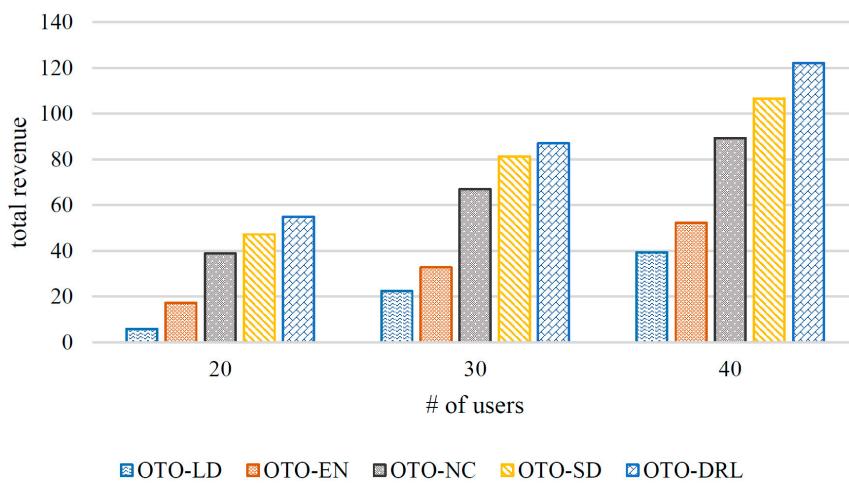
**Figure 4.** Convergence of total revenue for 30 users.



**Figure 5.** Convergence of total revenue for 40 users.



**Figure 6.** The user activity trajectory is the total revenue under ten steps.



**Figure 7.** The user activity tracking is the total revenue under 20 steps.

The experimental results lead to the following conclusions: (1) For a group of users with the same mobile trajectory, the total revenue of the OTO-DRL algorithm is significantly higher than that of the other four comparison algorithms in the decision-making process. As shown in Figure 5, the blue and yellow lines represent the total revenue results of OTO-LD and OTO-EN, respectively. Compared with the initial decision, the results of OTO-EN are slightly higher than those of OTO-DRL in the initial iteration, which is related to the size of the user data and the service's configuration file. The performance of OTO-EN and OTO-LD is affected by communication overhead and migration delays. In our experiments, users were set to send data packets continuously at equal time intervals. Therefore, communication delays will increase when users move frequently, and extensive delays may occur under OTO-EN. (2) The increase in the number of users specifically impacts convergence. As shown in Figures 5–7, the convergence speed slows down as the number of users increases. As shown in Figures 5 and 6, after 100 iterations, the total revenue of the users approaches convergence. However, as shown in Figure 7, the total revenue of 40 users approaches convergence after 200 iterations. The reason is that the increase in the number of users means a corresponding increase in the number of services, and the probability of conflicts during the terminal task offloading process will increase, which may slow down the convergence speed. (3) For each group of users, the overall revenue fluctuates within a relatively fixed range. Due to the relatively dense supply of edge servers, many overlapping areas provide users with multiple choices. During the learning process of OTO-DRL, due to different offloading locations, the total revenue

generated by these results will fluctuate between several relatively fixed values during the convergence process, mainly due to the deviation of user activity trajectories and task offloading locations. The fluctuation of OTO-NC and OTO-SD strategies is significantly different, and the overall revenue strategy of OTO-SD is better than that of OTO-NC. The main reason is that once the edge node has insufficient resources in the selection process of OTO-NC, the terminal user load will increase sharply.

### 5.3. Performance Evaluation

Based on the above analysis of the convergence of different groups of users, we evaluated the overall revenue when the user activity trajectories differed between the two groups in different time series scenarios. The results are shown in Figures 6 and 7. In addition, we obtained the following observations from the analysis: (1) The number of user activity trajectories generated in different periods will affect the overall revenue. As shown in Figure 7, when the user trajectory is ten steps, the highest total revenue of different groups of users under the four strategies is much lower than in the case of 20 trajectories. (2) The erratic activities of end users make the overall benefits under these four algorithms completely different. For users with 10 trajectories, the total benefit of the groups of 20 and 30 using the OTO-DRL strategy is higher than that of OTO-SD, but the overall difference is insignificant. However, for the group of 40 users in the OTO-DRL strategy, the gap between the overall income and the comparison algorithm is significant. At the same time, with the increase in the number of users, the revenue of the OTO-NC strategy decreases more than that of the OTO-SD strategy. For users with 20 trajectories, the total income of these three groups under OTO-DRL is higher than that of the comparison strategy. However, it is evident that with the increase in trajectories, the income gap between OTO-NC and OTO-SD strategies tends to narrow. In conclusion, OTO-NC performs better among users of different scales in mobile edge computing.

## 6. Conclusions

In this section, we study the problem of online task offloading in edge collaboration scenarios. We propose a reward evaluation method based on deep reinforcement learning to offload online tasks for multiple mobile users and jointly optimize the overall delay and energy consumption under the constraints of edge physical resources. On this basis, aiming at the characteristics of user mobility, a task optimization scheduling mechanism based on the multi-dimensional characteristics of tasks and edge resource conditions is proposed to avoid invalid states in the decision-making module and approximate strategies in the decision-making module according to the introduced feasibility mechanism. Finally, we conduct experiments on synthetic simulation datasets and real datasets, respectively. We provide corresponding conclusions in the synthetic simulation dataset section by analyzing and evaluating the proposed joint optimization method with several state-of-the-art methods. The experimental results show that the scheme can well realize the increase in the number of tasks requested by mobile terminal users in the edge collaborative service area while providing a guarantee for the service quality of task requests with higher priority.

**Author Contributions:** Conceptualization, M.S. and T.B.; methodology, M.S.; software, M.S.; validation, M.S., T.B. and D.X.; formal analysis, M.S.; investigation, M.S.; resources, H.L.; data curation, H.L. and G.S.; writing—original draft preparation, M.S.; writing—review and editing, D.X. and H.L.; visualization, M.S.; supervision, T.B. and G.S.; project administration, T.B.; funding acquisition, T.B. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [[CrossRef](#)]
2. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [[CrossRef](#)]
3. Cisco, U. Cisco annual internet report (2018–2023) white paper. *Cisco San Jose CA USA* **2020**, *10*, 1–35.
4. Stryjak, J. *The Mobile Economy 2020*; GSMA: London, UK, 2020.
5. Quan, T.; Zhang, H.; Yu, Y.; Tang, Y.; Liu, F.; Hao, H. Seismic Data Query Algorithm Based on Edge Computing. *Electronics* **2023**, *12*, 2728. [[CrossRef](#)]
6. Mukherjee, M.; Kumar, S.; Mavromoustakis, C.X.; Mastorakis, G.; Matam, R.; Kumar, V.; Zhang, Q. Latency-driven parallel task data offloading in fog computing networks for industrial applications. *IEEE Trans. Ind. Inform.* **2019**, *16*, 6050–6058. [[CrossRef](#)]
7. Sarkar, I.; Adhikari, M.; Kumar, N.; Kumar, S. Dynamic task placement for deadline-aware IoT applications in federated fog networks. *IEEE Internet Things J.* **2021**, *9*, 1469–1478. [[CrossRef](#)]
8. Yang, B.; Cao, X.; Bassey, J.; Li, X.; Kroecker, T.; Qian, L. Computation Offloading in Multi-Access Edge Computing Networks: A Multi-Task Learning Approach. In Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC), Shanghai, China, 21–23 May 2019; pp. 1–6.
9. Tong, Z.; Deng, X.; Ye, F.; Basodi, S.; Xiao, X.; Pan, Y. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Inf. Sci.* **2020**, *537*, 116–131. [[CrossRef](#)]
10. Yuan, Q.; Li, J.; Zhou, H.; Lin, T.; Luo, G.; Shen, X. A joint service migration and mobility optimization approach for vehicular edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 9041–9052. [[CrossRef](#)]
11. Zhan, W.; Luo, C.; Min, G.; Wang, C.; Zhu, Q.; Duan, H. Mobility-aware multi-user offloading optimization for mobile edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 3341–3356. [[CrossRef](#)]
12. Zhou, J.; Tian, D.; Wang, Y.; Sheng, Z.; Duan, X.; Leung, V.C. Reliability-oriented optimization of computation offloading for cooperative vehicle-infrastructure systems. *IEEE Signal Process. Lett.* **2018**, *26*, 104–108. [[CrossRef](#)]
13. Zhang, H.; Guo, J.; Yang, L.; Li, X.; Ji, H. Computation Offloading Considering Fronthaul and Backhaul in Small-Cell Networks Integrated with MEC. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017; pp. 115–120.
14. Xu, J.; Li, X.; Ding, R.; Liu, X. Energy Efficient Multi-Resource Computation Offloading Strategy in Mobile Edge Computing. 2019. Available online: [https://dro.deakin.edu.au/articles/journal\\_contribution/Energy\\_efficient\\_multi-resource\\_computation\\_offloading\\_strategy\\_in\\_mobile\\_edge\\_computing/20742406c](https://dro.deakin.edu.au/articles/journal_contribution/Energy_efficient_multi-resource_computation_offloading_strategy_in_mobile_edge_computing/20742406c) (accessed on 1 January 2019).
15. Ouyang, T.; Zhou, Z.; Chen, X. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 2333–2345. [[CrossRef](#)]
16. Wei, F.; Chen, S.; Zou, W. A greedy algorithm for task offloading in mobile edge computing system. *China Commun.* **2018**, *15*, 149–157. [[CrossRef](#)]
17. Zhang, H.; Wu, W.; Wang, C.; Li, M.; Yang, R. Deep Reinforcement Learning-Based Offloading Decision Optimization in Mobile Edge Computing. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019; pp. 1–7.
18. Lu, S.; Wu, J.; Duan, Y.; Wang, N.; Fang, J. Towards cost-efficient resource provisioning with multiple mobile users in fog computing. *J. Parallel Distrib. Comput.* **2020**, *146*, 96–106. [[CrossRef](#)]
19. Yu, B.; Pu, L.; Xie, Y.; Jian, Z. Joint task offloading and base station association in mobile edge computing. *J. Comput. Res. Dev.* **2018**, *55*, 537–550.
20. Chang, J.; Kang, M.; Park, D. Low-power on-chip implementation of enhanced svm algorithm for sensors fusion-based activity classification in lightweighted edge devices. *Electronics* **2022**, *11*, 139. [[CrossRef](#)]
21. Liu, L.; Liu, X.; Zeng, S. Research on virtual machines migration strategy based on mobile user mobility in mobile edge computing. *J. Chongqing Univ. Posts Telecommun. Nat. Sci. Ed.* **2019**, *18*, 570–584.
22. Gao, L.; Moh, M. Joint Computation Offloading and Prioritized Scheduling in Mobile Edge Computing. In Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS), Orleans, France, 16–18 July 2018; pp. 1000–1007.
23. Kim, T.; Sathyaranayana, S.D.; Chen, S.; Im, Y.; Zhang, X.; Ha, S.; Joe-Wong, C. Modems: Optimizing edge computing migrations for user mobility. *IEEE J. Sel. Areas Commun.* **2022**, *41*, 675–689. [[CrossRef](#)]
24. Ale, L.; Zhang, N.; Fang, X.; Chen, X.; Wu, S.; Li, L. Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 881–892. [[CrossRef](#)]
25. Hazra, A.; Donta, P.K.; Amgoth, T.; Dustdar, S. Cooperative transmission scheduling and computation offloading with collaboration of fog and cloud for industrial IoT applications. *IEEE Internet Things J.* **2022**, *10*, 3944–3953. [[CrossRef](#)]
26. Wu, T.; Jiang, M.; Han, Y.; Yuan, Z.; Li, X.; Zhang, L. A traffic-aware federated imitation learning framework for motion control at unsignalized intersections with internet of vehicles. *Electronics* **2021**, *10*, 3050. [[CrossRef](#)]
27. Ding, Y.; Liu, C.; Zhou, X.; Liu, Z.; Tang, Z. A code-oriented partitioning computation offloading strategy for multiple users and multiple mobile edge computing servers. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4800–4810. [[CrossRef](#)]
28. Wei, X.; Wang, Y. Joint Resource Placement and Task Dispatching in Mobile Edge Computing across Timescales. In Proceedings of the 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), Tokyo, Japan, 25–28 June 2021; pp. 1–6.

29. Wu, G.; Li, Z. Task Offloading Strategy and Simulation Platform Construction in Multi-User Edge Computing Scenario. *Electronics* **2021**, *10*, 3038. [[CrossRef](#)]
30. Guo, F.; Zhang, H.; Ji, H.; Li, X.; Leung, V.C. An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2651–2664. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.