future internet

Article

Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT

Youpeng Tu [1], Haiming Chen [1,2,*], Linjie Yan [1] and Xinyan Zhou [1]

[1] Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China; 1911082218@nbu.edu.cn (Y.T.); yan.linjie@foxmail.com (L.Y.); zhouxinyan@nbu.edu.cn (X.Z.)
[2] Zhejiang Provincial Key Laboratory of Mobile Network Application Technology, Ningbo University, Ningbo 315211, China
* Correspondence: chenhaiming@nbu.edu.cn

Abstract: In IoT (Internet of Things) edge computing, task offloading can lead to additional transmission delays and transmission energy consumption. To reduce the cost of resources required for task offloading and improve the utilization of server resources, in this paper, we model the task offloading problem as a joint decision making problem for cost minimization, which integrates the processing latency, processing energy consumption, and the task throw rate of latency-sensitive tasks. The Online Predictive Offloading (OPO) algorithm based on Deep Reinforcement Learning (DRL) and Long Short-Term Memory (LSTM) networks is proposed to solve the above task offloading decision problem. In the training phase of the model, this algorithm predicts the load of the edge server in real-time with the LSTM algorithm, which effectively improves the convergence accuracy and convergence speed of the DRL algorithm in the offloading process. In the testing phase, the LSTM network is used to predict the characteristics of the next task, and then the computational resources are allocated for the task in advance by the DRL decision model, thus further reducing the response delay of the task and enhancing the offloading performance of the system. The experimental evaluation shows that this algorithm can effectively reduce the average latency by 6.25%, the offloading cost by 25.6%, and the task throw rate by 31.7%.

Keywords: computational offloading; resource allocation; prediction; DRL; LSTM

check for updates

Citation: Tu, Y.; Chen, H.; Yan, L.; Zhou, X. Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT. *Future Internet* **2022**, *14*, 30. https://doi.org/10.3390/fi14020030

Academic Editors: Paolo Bellavista and Vijayakumar Varadarajan

Received: 1 December 2021
Accepted: 16 January 2022
Published: 18 January 2022

1. Introduction

With the rapid development of IoT and the exponential growth of device scale, traditional cloud computing [1] can no longer meet the service demand of IoT devices. In this context, edge computing [2] technology has emerged. The essence of edge computing is to place some small servers at the edge of the network, which is closer to the location of user devices, and its location is between cloud servers and terminal devices. The user can offload tasks that were originally offloaded to the cloud server to the edge server for executing some tasks, thus reducing the delay in task delivery and improving the quality of service of the tasks [3].

The core of edge computing is computation offloading [4], that is, offloading tasks from terminal devices to edge servers to reduce the computational workload of terminal devices and reduce the computational energy consumption of devices to extend their operating time. Specifically, the offloading problem consists of two aspects: (1) whether the tasks need to be offloaded; and (2) to which edge server the tasks are offloaded. Although the computational offloading problem has been well studied in the cloud computing area, the computational offloading of tasks still faces considerable challenges in the emerging edge computing scenario. For example, additional transmission delays and energy consumption are incurred when the number of offloading tasks is large, or in complex scenarios with multiple terminal devices and multiple edges, the edge system can suffer from load imbalance and affect the offloading performance if there is no reasonable mechanism of resource allocation [5].

At present, researchers have conducted a lot of research on the offloading decision problem, such as seeking the optimal solution after formulating the problem based on traditional algorithms [6–10], maximizing the reward to make appropriate decision actions based on reinforcement learning algorithms [11–14], etc. However, the above algorithms do not take the dynamic and predictable nature of the task during task offloading into account, thus they may produce unreasonable offloading decisions, resulting in degradation of the task offloading performance. Traditional task offloading methods can only optimize task offloading and resource allocation statically. Although deep reinforcement learning methods can meet the requirement of current IoT dynamic task computing, most algorithms only make the model converge faster and achieve better results by improving the method during model training, while ignoring the optimization of the task inference testing process, which still brings relatively high response delays in real scenarios. In other words, the generation and processing of tasks in real scenarios are more correlated with time series [15,16], so the prediction can be made by characterizing the tasks. The load of each server can be predicted by collecting the historical data of the system [17,18], and using the prediction method can efficiently allocate resources and make offload decisions for tasks with limited resources, which can effectively reduce the response delay of tasks when making the decision. To address the above problems and challenges encountered in task offloading, this paper proposes an Online Predictive Offloading (OPO) algorithm. The algorithm combines Long Short-Term Memory (LSTM) and deep reinforcement learning (DRL) to predict task dynamic information in real-time, based on the observed edge network condition and the server load. This algorithm aims to make offloading decisions by taking into account the task processing delays, the task tolerance delays, and the task computation energy consumption, and to avoid causing network congestion and server overloading, thus minimizing the task dropped rate and reduced the computational cost of the task. The main contributions of this paper are summarized as follows.

- Modeling the problem of computing offloading in a multi-edge, multi-device computing scenario as a nonlinear optimization problem. Moreover, the goal of task offloading is minimizing long-term costs in terms of latency and energy consumption.
- By predicting the characteristics of tasks and edge server loads, tasks are dynamically offloaded to the optimal edge server. In the decision model, the prediction is combined with task decision to dynamically allocate resources for different tasks to further reduce latency and improve service quality.
- The proposed model and method are extensively evaluated with real-world datasets. The results reveal that the model developed in this paper can effectively reduce the cost using the DRL algorithm with Deep Q Network (DQN) and its variants. The OPO algorithm can maintain low task latency and task discard rate when facing large and complex scenarios.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 gives the system model and the formulation of the problem, Section 4 gives the framework of the model in the training and offloading phases and designs the reinforcement learning offloading algorithm for the given objective. Section 5 presents the experimental evaluation of the proposed method. Section 6 concludes the work of this paper.

2. Related Works

The purpose of the computing offloading is to make the best decision for the task to improve Quality of Service (QoS), which can indirectly enhance the computing power of the terminal device. Since real scenarios are complex and dynamic, which implies that the system model and offloading algorithms should be sufficiently scalable. There is a lot of existing work on computing offloading, which can be classified based on the objects of the system model during the establishment phase or the different methods that are used in the proposed model.

2.1. *Offloading Methods with Different Modeling Objects*

According to the task type and offloading requirements, the offloading decisions are generally classified into three categories: minimizing delay, minimizing energy consumption, and minimizing system cost. Offloading strategies for minimizing delay are to complete the task execution with the lowest time requirement. Offloading strategies for minimizing energy consumption are to make the whole system with the lowest energy consumption while satisfying certain delay requirements during task execution. Offloading strategies for minimizing system cost are generally to make a trade-off between delay and energy consumption, particularly trying to find a balance between energy consumption and delay to meet the different user requirements of IoT applications.

Shu et al. [19] studied the fine-grained task offloading problem in edge computing for low power IoT systems and proposed a lightweight and effective offloading scheme for multi-user edge systems to minimize the execution time by offloading the most appropriate IoT tasks to the edge servers. Guo et al. [20] consider the scenario where multiple terminal users offload repetitive computational tasks to network edge servers and share computational results among these servers. They designed optimal fine-grained collaborative offloading strategies that utilize data caching to minimize task execution delays at the terminal. Ali et al. [21] proposed a federated RL-based channel resource allocation framework for the fifth generation (5G) networks, which suggested collaborating learning estimates for faster learning convergence. Ali et al. [22] made extensive research efforts on developing beyond sixth generation (6G) wireless networks, which aimed at bringing ultra-reliable low-latency communication services. Zhao et al. [23] found it difficult to achieve a balance between high resource consumption and high communication costs and proposed a local computing offloading method that minimizes the total energy consumption consumed by the terminal devices and edge servers by jointly optimizing the task offloading rate, the CPU frequency of the system, the allocated bandwidth of the available channels, and the transmission power of each device at each time slot. Vu et al. [24] proposed an edge computing network architecture that enables edge nodes to share computational and radio resources, to minimize the total energy consumption of terminal users while satisfying the latency requirements of the tasks. Yuan et al. [25] proposed a cloud-edge computing system that includes the terminal layer, edge layer, and cloud layer. Based on this, a profit-maximizing collaborative computing offloading and resource allocation algorithm is designed to minimize the system cost and ensure that the response time of the tasks is satisfied. Alqerm et al. [26] addressed the resource allocation problem in Edge-IoT systems through developing a novel framework named DeepEdge, which allocated resources to the heterogeneous IoT applications with the goal of maximizing users' Quality of Experience (QoE).

2.2. *Offloading Methods with Different Problem Solving Strategies*

In problem solving, some scholars use traditional methods. Thai et al. [6] proposed a generic architecture for cloud edge computing with both vertical and horizontal service computing nodes, formulated as a mixed-integer nonlinear programming problem, proposed an approximation algorithm to iteratively obtain the optimal solution using a branch-and-bound method to minimize the system computation and communication costs. Cui et al. [7] made a trade-off between energy consumption and latency to meet the user requirements of different IoT applications, formalized the problem as a constrained multi-objective optimization problem, and found the optimal solution by an improved fast elite non-dominated ranking genetic algorithm. Gu et al. [8–10] solved the joint optimization problem with nonconvexity after transforming it into a convex optimization problem.

With the increasing number of terminal devices and edge devices, the heterogeneity of the network becomes complex. At the same time, Social Internet of Things [27] and massive Internet of Things were derived from the traditional Internet of Things. The incumbent Internet of Things suffered from poor scalability and elasticity in communication, computing, caching and control problems. The recent advances in DRL algorithms can potentially address the above problems of IoT systems. Chen et al. [28] did a comprehensive survey and provided a state-of-the-art literature review on a wide variety of IoT applications enabled by DRL algorithms. Therefore, some researchers combine neural networks and reinforcement learning in dealing with task offloading problems to explore unknown complex dynamic IoT environment information to make decisions.

Huang et al. [29] proposed a distributed algorithm based on deep learning into the advantage of multiple parallel neural networks to generate optimal solutions without manual labeling of data. Tang et al. [30] formulated a task offloading problem to minimize the long-term resource consumption and proposed a reinforcement learning distributed algorithm where each device can give task decisions without knowing the offloading decisions of other devices. Jang et al. [31] proposed a method, making it possible to simultaneously perform knowledge transfer and policy model compression in a single training process on edge devices with considered their limited resource budgets. The training time elapsed for edge policy training with this method is reduced significantly compared with edge policy training from scratch. Gong et al. [32] defined a total cost function as the weighted sum of task delay and energy consumption, and they transform the initial problem into a convex optimization problem, which was solved by an atom action generation technique and adaptive aggregation parameter update strategy. Chen et al. [33] considered the communication of terminal devices in an ultradense LAN, where users can select multiple base stations for task offloading to maximize utility performance, reducing the energy consumption of tasks in the computational queue and the channel queue between terminals and base stations, breaking the bottleneck in the high-dimensional space, and achieving the optimal policy based on the DQN algorithm in the case of unknown network dynamic environment. Similarly, the design goal of Song et al. [34] is to optimize resource utilization, energy consumption, and network latency by predicting user behavior using a variant of the DQN algorithm to solve the problem. Zou et al. [35–37] both used the improved Actor-Critic (AC) algorithm in the Deep Deterministic Policy Gradient (DDPG) algorithm to solve the offloading decision policy to balance the workload of the edge server and the final task was reduced in terms of energy consumption and computation time. Chen et al. [38,39] based on Monte Carlo Tree Search (MCTS) and Deep Neural Network (DNN) algorithm, the proposed algorithm spreads the high-dimensional decision space to each layer of the Monte Carlo tree and makes the policy search tend to the space with higher reward, which improves the search efficiency and reduces the scope of the search space.

3. System Model

The model in this paper is built on a multi-terminal, multi-edge network scenario, in which the set of terminal layer devices are denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$. On each MD (Mobile device), there exists a task queue and a computation queue, where the task queue stores the tasks to be decided for offloading and the computation queue processes the tasks that are executed locally. Additionally, the set of edge layer servers are denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. Multiple computation queues are included in each edge server for parallel computation of transmission queue offload tasks. Figure 1 shows an illustration of EC system with a mobile device and an edge node. In the following, we first present the task model and the task offloading decision, respectively. Then, we introduce the computation, communication model and prediction model.

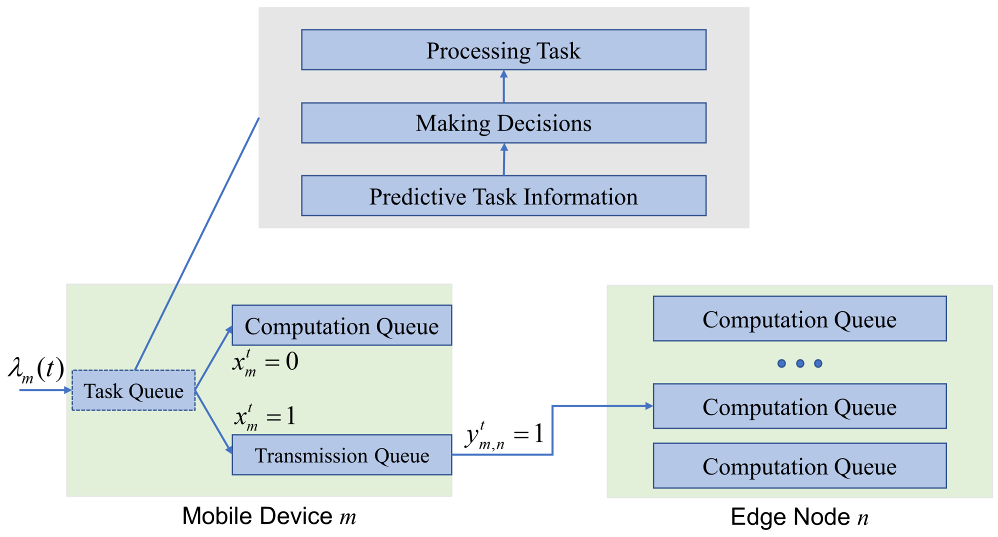


Figure 1. An illustration of EC system with a mobile device and an edge node.

3.1. *Task Model*

For any MD, the tasks generated in different time slots are identified by $\mathcal{T} = \{1, 2, \ldots, T\}$. The arrival time and data size of the tasks are used from real-world collected IoT data. For each arriving task, it is first stored in the corresponding MD task cache queue, and then the decision model gives where the task will be offloaded to be executed. For $t \in \mathcal{T}$ time slot, the new task generated by the terminal device $m \in \mathcal{M}$ is denoted as $\lambda_m(t) = \left(D_m^t, \rho_m^t, \tau_{m,\max}^t\right)$. D_m^t denotes the size of the task data size, ρ_m^t denotes the computational resources required per bit for this type of task, and $\tau_{m,\max}^t$ denotes the maximum tolerated delay of the task, i.e., the new task generated in time slot t will be thrown when the task has not been completed in time slot $t + \tau_{m,\max}^t - 1$.

3.2. *Decision Model*

When the terminal device m has a new task $\lambda_m(t)$ in time slot t, the decision model has to give the offloading scheme. The binary variable $x_m^t \in \{0, 1\}$ is used to indicate whether the current task is offloaded, $x_m^t = 0$ indicates that the task is executed on the MD, and $x_m^t = 1$ indicates that the task will be offloaded to an edge server for execution. $y_{m,n}^t \in \{0, 1\}$ represents the edge server to which the task is offloaded for execution, when $y_{m,n}^t = 1$, the task is offloaded to the edge server $n \in \mathcal{N}$ for execution, otherwise $y_{m,n}^t = 0$. The tasks considered in this model are atomic level tasks, i.e., the tasks are not subdividable, each offloaded task can be executed in only one edge server, and the tasks offloaded to the edge server for execution are constrained by $\sum_{n\in\mathcal{N}} y_{m,n}^t = 1, m \in \mathcal{M}, t \in \mathcal{T}, x_m^t = 1$.

3.3. *Computational Model*

3.3.1. Terminal Layer Computing Model

In the MD model, two kinds of queues are included: the task queue, in which newly generated tasks but not yet assigned decisions are stored, and the task computation queue, which is used to output the computation results of tasks. It is specified that only one task can enter the queue for processing in a time slot.

The task generated by the t time slot must wait until the computation queue is free to execute the computation, i.e., wait for the previous task to complete its execution, and its waiting delay $\tau_{m,\text{wait}}^t$ can be expressed by Equation (1).

$$\tau_{m,\text{ wait}}^t = \left[\max_{t'\in\{0,1,\ldots,t-1\}} l_m^{\text{comp}}\left(t'\right) - t + 1\right]^+ \tag{1}$$

where $l_m^{\text{comp}}(t)$ denotes the completion time slot of the task (completing the task or being discarded). When the task is processed in the computation queue, there is a processing delay. Suppose that the processing capacity (bits/s) of the MD is f_m^{device}, then the processing delay $\tau_{m,exe}^t$ of the task in the computational queue can be expressed by Equation (2).

$$\tau_{m,\text{ exe}}^t = \frac{D_m^t \rho_m^t}{f_m^{\text{device}}} \tag{2}$$

Therefore, we can find the completion time slot $l_m^{\text{comp}}(t)$ of task $\lambda_m(t)$ by the above equation as shown in Equation (3).

$$l_m^{\text{comp}}(t) = \min\{t + \tau_{m,\text{ wait}}^t + \tau_{m,\text{ exe}}^t, \quad t + \tau_{m,\max}^t\} \tag{3}$$

$t + \tau_{m,\text{ wait}}^t + \tau_{m,\text{ exe}}^t$ denotes the time slot when the task execution is completed and $t + \tau_{m,\max}^t$ denotes the maximum tolerated time slot of the task. The total processing delay $\tau_{m,MD}^t$ of task $\lambda_m(t)$ on the MD can be expressed by Equation (4).

$$\tau_{m,MD}^t = \min\{\tau_{m,\text{ wait}}^t + \tau_{m,\text{ exe}}^t, \quad \tau_{m,\max}^t\} \tag{4}$$

At the same time, this paper considers the energy consumption of MD in processing tasks, which consists of two parts: (1) task computation energy consumption expressed as the product of computation power and computation time, and (2) task waiting energy consumption, expressed as the product of waiting power and time. Therefore, the energy consumption E_m^{device} required for the task to be executed locally is shown as follows.

$$E_m^{\text{device}} = P_m^{\text{exe}} \tau_{m,\text{ exe}}^t + P_m^{\text{wait}} \tau_{m,\text{ wait}}^t \tag{5}$$

3.3.2. Edge Layer Computing Model

Since the computational resources of the edge server are much larger than those of the MD, the task can be immediately entered into the computational queue for execution when the task offloaded by the MD arrives, so the waiting delay of the task is not considered on the edge server. For the task $\lambda_m(t)$ generated by device m in time slot t, it is denoted by $\lambda_{m,n}(t)$ on the edge server. Assume that the current processing capacity (bits/s) allocated by the edge node for processing the task is f_n^{EC}. Then, the processing latency $\tau_{m,n,\text{ exe}}^t$ of the task at the edge layer can be expressed as:

$$\tau_{m,n,\text{ exe}}^t = \frac{D_m^t \rho_m^t}{f_n^{EC}} \tag{6}$$

Similar to terminal layer computing, the completion time slot $l_{m,n}^{\text{comp}}(t)$ for edge layer tasks $\lambda_{m,n}(t)$ can be expressed as:

$$l_{m,n}^{\text{comp}}(t) = \min\{t + \tau_{m,\text{ wait}}^t + \tau_{m,n}^{\text{tran}} + \tau_{m,n,\text{ exe}}^t, \quad t + \tau_{m,\max}^t\} \tag{7}$$

The total delay $\tau_{m,nEC}^t$ of task $\lambda_{m,n}(t)$ on edge server n can be expressed as

$$\tau_{m,nEC}^t = \min\{\tau_{m,\text{ wait}}^t + \tau_{m,n}^{\text{tran}} + \tau_{m,n,\text{ exe}}^t, \quad \tau_{m,\max}^t\} \tag{8}$$

where $\tau_{m,\text{ wait}}^t$ denotes the waiting delay in the local model, $\tau_{m,n}^{\text{tran}}$ denotes the transmission delay, $\tau_{m,\text{ wait}}^t + t_{m,n}^{\text{tran}} + \tau_{m,n,\text{ exe}}^t$ denotes the time slot required for a task to be offloaded from the endpoint to the edge server and executed to completion, and $\tau_{m,\max}^t$ denotes the maximum tolerated delay.

The energy consumption $E_{m,n}^{\text{edge}}$ incurred when tasks are offloaded to the edge server for execution can be expressed as

$$E_{m,n}^{\text{edge}} = P_m^{\text{wait}} \tau_{m,\text{ wait}}^t + P_{m,n}^{\text{tran}} \tau_{m,n}^{\text{tran}} + P_{m,n}^{\text{exe}} \tau_{m,n,\text{ exe}}^t \tag{9}$$

where $P_m^{\text{wait}} \tau_{m,\text{ wait}}^t$,$P_{m,n}^{\text{tran}} \tau_{m,n}^{\text{tran}}$,$P_{m,n}^{\text{exe}} \tau_{m,n,\text{ exe}}^t$, respectively, denote the waiting energy consumption, transmission consumption, and edge node computation consumption of the task, respectively.

3.4. Communication Model

Tasks consume bandwidth resources and incur transmission delays only when they are offloaded to the edge server for execution. The offloading of the MDs task at time slot t is $x_{\text{all}}^t = \left[x_1^t, x_2^t, x_3^t \cdots x_m^t\right]$, for the bandwidth occupied when offloading the task is $B_{\text{all}}^t = \left[B_1^t, B_2^t, B_3^t \cdots B_m^t\right]^{\text{T}}$, and the task is subject to the bandwidth constraint when offloading, i.e., $x_{\text{all}}^t \cdot B_{\text{all}}^t \leq B_{\max}$. According to Shannon's formula, we can find the transmission rate.

$$r_{m,n} = B_m^t \log_2\left(1 + \frac{P_{m,n}^{\text{tran}} g}{\sigma^2}\right) \tag{10}$$

where g is the channel gain and σ^2 denotes the power of additive Gaussian white noise. Thus, the transmission delay and transmission energy consumption of the task can be derived from Equations (11) and (12).

$$\tau_{m,n}^{\text{tran}} = \frac{D_m(t)}{r_{m,n}} \tag{11}$$

$$E_{m,n}^{\text{tran}} = P_{m,n}^{\text{tran}} \tau_{m,n}^{\text{tran}} \tag{12}$$

In summary, the overall model of the system is a trade-off between the time delay and energy consumption of the task computation to create a minimization cost problem (13), and the solution goal is to minimize the total cost of the tasks generated in the system over time.

$$\begin{aligned}
&\mathbb{C} = \sum_{n=1}^{N} \sum_{m=1}^{M} \Big(\alpha\big(\tau_{m,MD}^t + \tau_{m,nEC}^t\big) + \beta\Big((1 - x_m(t))E_m^{\text{device}} + x_m(t) y_{m,n}(t) E_{m,n}^{edge}\Big)\Big) \\
&\min \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{C} \\
&s.t. \quad \text{C1} : x_m^t \in \{0,1\} \\
&\text{C2} : y_{m,n}^t \in \{0,1\} \\
&\text{C3} : \sum_{n\in\mathcal{N}} y_{m,n}^t = 1, \forall m \in \mathcal{M}, t \in \mathcal{T}, x_m^t = 1 \\
&\text{C4} : \tau_{m,MD}^t, \tau_{m,nEC}^t \leq \tau_{m,\max}^t \\
&\text{C5} : f_m^{\text{device}} \leq f_{\max}^{\text{device}} \\
&\text{C6} : \sum_{n\in\mathcal{N}} f_n^{EC} \leq f_{\max}^{EC} \\
&\text{C7} : 0 \leq P_{m,n}^{\text{tran}} \leq P_{\max} \\
&\text{C8} : \sum_{n\in\mathcal{M}} x_{\text{all}}^t \cdot B_{\text{all}}^t \leq B_{\max}, \forall t \in \mathcal{T}
\end{aligned} \tag{13}$$

The constraints C1–C3 indicate that the generated task can only select one computing node for computational processing; C4 indicates that the total processing delay of the task, whether it is processed locally or offloaded to an edge node for processing, should not exceed the maximum tolerated delay of the task; C5–C6 indicate that the computational resources allocated for the task are not allowed to exceed the maximum processing capacity of the node; C7–C8 indicate that the transmission power of the task cannot exceed the maximum transmission power, and the bandwidth occupied by all offloaded tasks in the same time slot should not exceed the maximum bandwidth resource. Table 1 lists the important notations used in this paper.

Table 1. Table of main notations.

Notation	Definition
$\mathcal{M}$	Set of terminal layer devices
$\mathcal{N}$	Set of edge layer servers
$\mathcal{T}$	Set of time slots for task generation
$\lambda_m(t)$	Task generated by terminal device m at time slot t
$\lambda_{m,n}(t)$	Task of device m are offloaded to edge node n
x_m^t	Device m offloading its task in time slot t while $x_m^t = 1$, Otherwise, $x_m^t = 0$
$y_{m,n}^t$	The task is offloaded to the edge server while $y_{m,n}^t = 1$, Otherwise, $y_{m,n}^t = 0$
$\tau_{m,\text{ wait}}^t$	Terminal layer device waiting delay in the task queue
$\tau_{m,\text{ exe}}^t$	Terminal layer device processing delay in the computation queue
$\tau_{m,\text{ MD}}^t$	Total time delay of the task generated by terminal devices m at time slot t
P_m^{exe}	Processing power of the task generated by terminal devices m at time slot t
P_m^{wait}	Waiting power of the terminal devices m
E_m^{device}	Energy consumption of the device m
$\tau_{m,n,\text{ exe}}^t$	Processing delay in the computation queue of the Edge server n
$\tau_{m,nEC}^t$	Total time delay of the edge server n
$P_{m,n}^{\text{tran}}$	Transmission power of the device m offload to edge server n
$\tau_{m,n}^{\text{tran}}$	Transmission delay of the device m offload to edge server n
$P_{m,n}^{\text{exe}}$	Processing power of the edge server n
$E_{m,n}^{\text{edge}}$	Energy consumption of the edge sever n
α, β	The trade off weight between energy consumption and delay in the system cost

3.5. Prediction Model

3.5.1. Task Prediction Model

In computational offloading systems, a decision process is required after task generation, and there will be a certain time delay from task generation to give a decision. Although task generation is a dynamic and random process, considering the long-term nature of the task, it will have a strong correlation with time.

Therefore, based on the history of user devices, we can predict the tasks that will be generated in the next network time slot of user devices, and load the service packages in advance before the real tasks are coming (e.g., allocate the best computation nodes in advance through the decision model). For example, for any MD with task data $D_t \in \{D_1, D_2, \cdots D_T\}$, taking $D_1, D_2, \cdots D_{t-1}$ as the input sequence, into the trained LSTM prediction model to predict the next time slot task data $D_t^{\sim}$. Therefore, the optimization goal of the prediction model is $|D_t - D_t^{\sim}| \propto 0$, i.e., to improve the accuracy of task data feature prediction as much as possible.

As shown in Figure 2, in a real scenario, we can predict the information of the future task by the prediction model, and determine the decision and allocate computing resources for the task. When the real task arrives, if the error between the real task and the predicted task is within the allowed threshold, the task is directly offloaded and computed according to the assigned decision information. Otherwise, the offloading decision is given using the decision model and the information of the new task is added to the historical data as training samples. By training the LSTM network, the weights and biases of each gate in the network are updated to improve the accuracy of the prediction model.

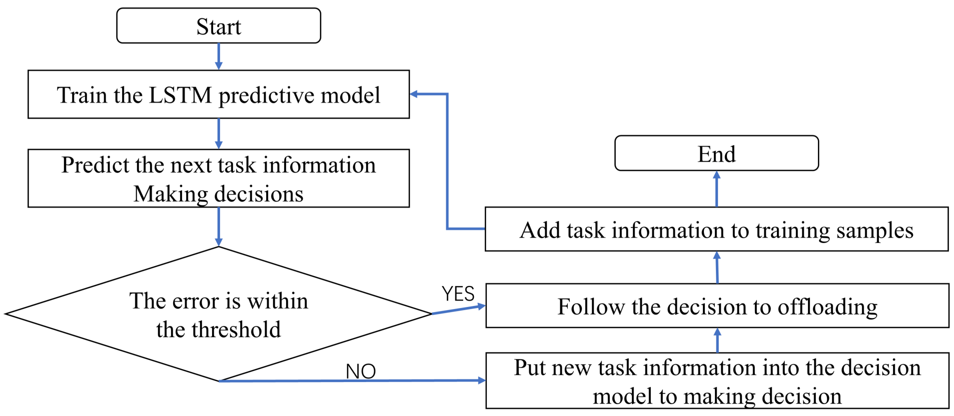

Start
Train the LSTM predictive model
Predict the next task information
Making decisions
The error is within the threshold
YES
NO
End
Add task information to training samples
Follow the decision to offloading
Put new task information into the decision model to making decision

Figure 2. Flow chart of task prediction.

3.5.2. Load Prediction Model

In an edge computing system, wrong offloading decisions may lead to load imbalance if the server load is not considered. The monitoring system on the edge server records system data and performance metrics, such as CPU occupancy, network usage, and current execution of tasks. We can use the logged data to predict the future server load. We refer to the load model in the literature [30] to predict the edge server load level (the number of idle queues on the edge nodes) using historical records. The output value $H_t^{\sim}$ is obtained from the trained LSTM load prediction model by using $H_1, H_2, \cdots H_{t-1}$ as the input sequence in the edge server historical load sequence $H_t \in \{H_1, H_2, \cdots H_T\}$. $h_t^{\sim} \in H_t^{\sim}$ is used as the predicted idle server, which can be preferentially selected as the offload computing node when training the DRL. As shown in Figure 3, in the DRL training process, the actions are selected by the agent through a certain probability, i.e., the action with the largest Q value is selected with ε probability, and the action is selected randomly with $1 - \varepsilon$ probability. In this paper, when the original agent selects a random action, the size comparison between a random value σ and the probability ε is used to determine whether it is a Random Action or a Prediction Action, so as to avoid falling into a local optimum. By giving the pre-selected action (idle server in the actual scenario) through Prediction Action, it can reduce the number of explorations by the agent and improve the convergence speed of the algorithm.

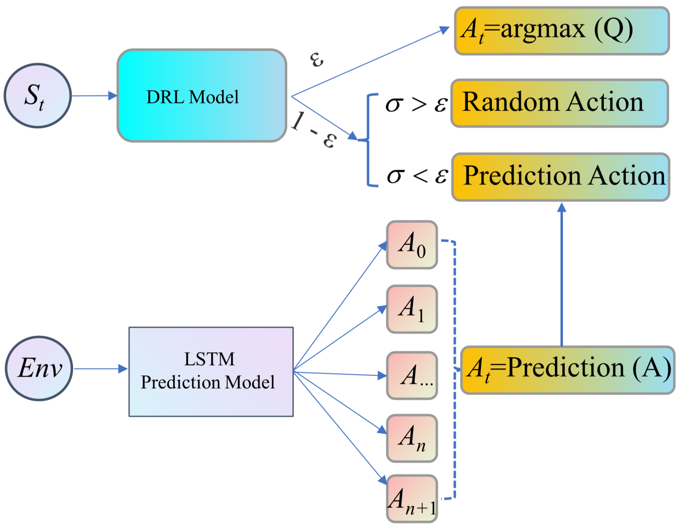


Figure 3. Illustration of offloading decision.

4. Model Solving

4.1. Overall Framework

In the model of DRL, the prediction model is combined with the deep reinforcement learning model. In this section, we introduce the model training phase, the model inference phase, and give the algorithm design for solving the model specifically.

4.1.1. Model Training Phase

The goal of DRL is to maximize the total reward by making the optimal action in each decision. DRL typically uses ε-greedy strategies for exploration and exploitation. Exploration is the random selection of any action with probability in expectation of a higher reward, otherwise, exploitation is the action with the largest action estimate. This stochastic strategy can fully explore the environment state, but it requires a long period of extensive exploration and low data utilization. In the model of this paper, action selection describes the offloading decision of the task, which simply means that the action space is known whether the task is to be executed locally or offloaded to one of the servers on the edge. When the agent performs stochastic exploration, as shown in Figure 4, this paper uses LSTM to predict the load of the edge server to give an optimal action, i.e., the optimal server at the next time slot is predicted by the historical load situation, and the offloading to the optimal server is done directly in this state to obtain a higher reward and also to effectively avoid the edge server load imbalance.

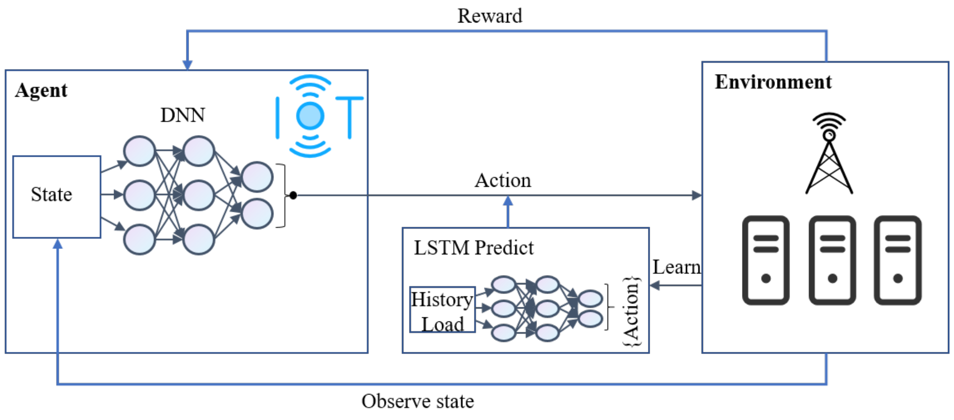


Figure 4. Illustration of model training phase.

4.1.2. Offloading Decision Phase

Each MD generates different types of tasks at different time slots. When a new task is generated, there is a response delay of the system to the task's decision request, i.e., the system gets the task information from the task queue before it can put the task information into the decision model to make a decision. For a task, there is a response delay of the system and a waiting delay in the queue between the generation of a task and giving a decision. As shown in Figure 5, the edge system can process the data from MD and store the processed records. Based on the historical records, the feature information of the next arriving task can be predicted by LSTM, and the predicted information will be given to the reinforcement learning decision model, which is proposed to make an offloading scheme for the predicted task. When the real task arrives, if the error between the real task and the predicted task is within the allowed range, the offloading decision of the task is given directly; otherwise, the decision is made according to the real task using the decision model. By predicting the information of the task, it can predict the target computation node of the task and effectively reduce the response and waiting delay of the task in the system.

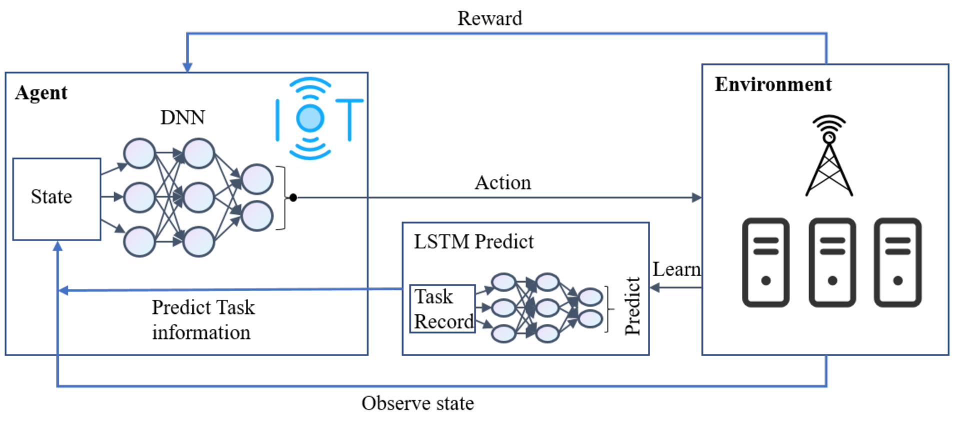


Figure 5. Illustration of offloading decision phase.

4.2. Algorithm Design

In this paper, we use the deep reinforcement learning algorithm DQN (Deep Q Network) and its variants based on the combination of reinforcement learning and deep learning to design a task offloading decision algorithm. A typical DQN model is composed of agent, state, action, and reward, and the policy is generated as a mapping $\pi : S \to A$ of states to actions to obtain a reward R, $r_t(s_t, a_t)$ denotes the reward that can be obtained by choosing action a_t in state s_t, and $R_0^{\gamma} = \sum_{(t=0)}^{T} \gamma^t r_t(s_t, a_t)$ is the long-term reward, where $T \to \infty, \gamma \in [0,1], r_t \in R, s_t \in S, a_t \in A$.

In traditional Q-learning, each state-action pair can be easily stored into the Q-table because the state space and action space are discrete and have small dimensions. However, when the state space and action space dimensions are large, as described in this paper, it is difficult to put all state-action pairs into Q-table. To solve this problem, the DQN model in DRL combines deep neural networks and Q-learning algorithms, and it transforms the Q-table tables into the Q-networks and uses neural networks to fit the optimal Q-functions. DQN relies mainly on the following key techniques.

Deep Q Network (DQN): There are two neural networks with the same structure but different parameters in DQN, i.e., the target network $\tilde{Q}(s', a'; \tilde{\theta})$and the main network $Q(s, a; \theta)$. When iteratively updating the network, the algorithm first uses the target network to generate the target Q-value as the label $f(t)$, and uses the loss function Loss($\boldsymbol{\theta}$) to update the parameters of the main network. After the introduction of the target network, the target Q value generated by the target network remains constant in time j, which can reduce the correlation between the current Q value and the target Q value and improve the stability of the algorithm. The target network will update the parameters $\tilde{\theta}$ with the Q network parameters θ every certain step.

$$f(t) = \begin{cases} r(t), \text{ if episode terminates at step } j+1 \\ r(t) + \gamma \max_{a'} \{\tilde{Q}(s', a'; \tilde{\theta})\}, \text{ otherwise} \end{cases} \tag{14}$$

$$\text{Loss}(\boldsymbol{\theta}) = (f(t) - Q(s, a; \theta))^2 \tag{15}$$

Replay Memory: In order to break the correlation within the data, DQN uses the experience replay method to solve this problem. After interacting with the environment, the agent is stored in the replay buffer in the form of (s_t, a_t, r_t, s_{t+1}). When executing valuation updates, the agent randomly selects a small set of experience tuples (s_t, a_t, r_t, s_{t+1}) from the replay buffer at each time step, and then the algorithm updates the network parameters by optimizing the loss function, using experience replay can not only make training more efficient, but also reduce the problem overfitting that generated by the training process.

Double DQN: Double DQN is proposed to solve the overestimation problem. DQN takes the maximum value with max each time, and the difference between this maximum value and the weighted average value introduces an error, which will lead to overestimation after a long time accumulation. The Double DQN is composed of two networks, Q^A and Q^B, and it utilizes these two networks to proceed the state valuation and the action output alternatively. That is, one network is used to select out the action, and the other network is used to update the Q value according to the selected action. The Double DQN makes the learning process more stable and reliable by separating the two steps of selecting the action corresponding to the Q value and evaluating the Q value corresponding to the action, which eliminates the overestimation brought by the greedy algorithm and obtains a more accurate Q estimation. Instead of finding the label value of parameter update directly from the target network, Double DQN finds the action corresponding to the maximum Q value in Q^A and then uses this selected action to compute the target value of parameter update in Q^B.

$$Q^A(s,a) \leftarrow Q^A(s,a) + \rho\left[r + \gamma \max_{a'} Q^B(s',a') - Q^A(s,a)\right] \tag{16}$$

$$Q^B(s,a) \leftarrow Q^B(s,a) + \rho\left[r + \gamma \max_{a'} Q^A(s',a') - Q^B(s,a)\right] \tag{17}$$

Dueling DQN: Compared with DQN, Dueling DQN considers the Q network into two parts, the first part is only related to the state S, and the specific action A to be adopted has nothing to do with this part is called the value function part, noted as $V^{\pi}(s)$, the second part is related to both the state S and action A, this part is called the action advantage function, noted as $A^{\pi}(s,a)$, the final value function can be expressed as

$$Q^{\pi}(s,a) = A^{\pi}(s,a) + V^{\pi}(s) \tag{18}$$

This equation cannot identify the role of $V^{\pi}(s)$ and $A^{\pi}(s,a)$ inside the final output. In order to reflect this identifiability, the action dominance function is generally to be the individual action dominance function minus the average of all action dominance functions in a state, which can ensure that the relative ordering of each action dominance function in that state remains unchanged, and can narrow the range of Q values, remove the redundant degrees of freedom, and improve the stability of the algorithm. The combination formula used in practice is as follows.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A\left(s,a';\theta,\alpha\right)\right) \tag{19}$$

4.2.1. Decision Model Elements

(1) Agent

In the model, each MD is considered as an agent (equivalent to a scheduler) that selects the next action according to the current state of the environment and improves the ability of the agent to make decisions by continuously interacting with the environment. The goal of the agent is to make the optimal action in any state, thus minimizing the total cost in the edge computing system.

(2) State

At the beginning of each time slot, each agent observes the state of the environment, which includes the properties of the MD task, the waiting queue state, the transmission queue state, bandwidth information, and the real-time load of the edge nodes, all the states are closely related to the action to be selected by the agent.

$$\boldsymbol{s}_m(t) = \left(\lambda_m(t), \tau^t_{m,\,\text{wait}}, \tau^{\text{tran}}_{m,n}, B^t, \boldsymbol{q}^{\text{edge}}_m(t-1), \boldsymbol{H}(t)\right) \tag{20}$$

where $\boldsymbol{H}(t)$ denotes the historical load level on each edge node and $\boldsymbol{q}^{\text{edge}}_m(t-1)$ is the edge node state information for time slot $t-1$.

(3) Action

Based on the current state, the agent first decides whether the newly generated task needs to be offloaded for computation, i.e., x_m^t; if it needs to be offloaded, it chooses which server to offload, i.e., y_m^t; and also chooses the appropriate transmission power when offloading the transmission, i.e., p_m^t.

$$\boldsymbol{a} = \left(x_m^t, y_m^t, p_m^t\right) \quad , \quad \mathcal{A} = \{0,1\}^{1+N+1} \tag{21}$$

(4) Reward

After observing the state at time slot t, the agent takes an action according to the policy and then receives a reward at time slot $t+1$ while updating the scheduling policy network to make an optimal decision in the next time slot. The goal of each agent is to maximize its long-term discounted reward by optimizing the mapping from states to actions so that the agent tends to make optimal decisions in its continuous interaction with the environment. The reward function is shown below, and the detailed design is described in Section 4.2.2.

$$\mathbb{E}\left[\sum_{(t=0)}^{T} \gamma^t r_t(s_t, a_t)\right] \tag{22}$$

4.2.2. Design of the Reward Function

The reward function is generally designed according to the objective function of the system model, which minimizes the total cost of processing delay and energy consumption by making optimal offloading decisions. In the model of this paper, tasks may exceed the limits of tolerated delay regardless of whether they are offloaded or not. These tasks are considered as unsuccessful tasks, but also consume the corresponding resources. In order to reduce the number of tasks thrown in the whole system and to minimize the cost of the system, a reasonable reward function must be designed.

For MDs that only consider saving local energy consumption, offloading is preferred. If all MDs offload the tasks, the edge server will be overloaded and the idle state of MDs is also a waste of computational resources, so the execution delay and energy consumption of tasks need to be considered at the same time. When a task completes successfully without exceeding a given tolerance delay, a corresponding reward is given; when a task exceeds its tolerance delay, the task is considered as discarded and penalized. With such a design, the action is rewarded whether the task is executed locally or on the edge server, as long as it is completed within the specified time. In this paper, for a task to be executed only if it is completed within the specified time and consumes the least amount of energy, the shorter the execution delay, the greater the reward the task receives, and the least cost of the whole system. In order to be consistent with the objective in the model of this paper, we use negative rewards. When the system objective function is at the minimum level, the DRL can get the maximum reward. Based on the above considerations, the reward function is designed as follows.

$$r_t(s_t, a_t) = -[(\alpha T + \beta E) - p_t] \tag{23}$$

where p_t is the penalty for the corresponding task dropped.

The structure of the DRL algorithm in this paper is based on the combination of DQN parameter update improved Double DQN and neural network structure improved Dueling DQN algorithm, and then the LSTM prediction algorithm is migrated to DRL decision algorithm. In this paper, we propose an Online Predictive Offloading (OPO) algorithm based on the deep reinforcement learning algorithm to solve the modeling problem, and the specific process of model training is shown in Algorithm 1.

In the algorithm, lines 8–14 with probability $1-\varepsilon$ select a random action or LSTM predict action a, and lines 21–25 are network parameter updates referencing the Double DQN update method.

Algorithm 1 Online Predictive Offloading Algorithm.

```
1:  Input: Input different tasks in each time solts
2:  Output: Optimal offloading decision and total cost
3:  Initialize $Q^A$,$Q^B$ and $s$
4:  Initialize replay memory D to capacity N;
5:  for episode = 1, M do
6:      Initialize sequence s, and preprocessed sequence
7:      for t = 1, T do
8:          With probability $1 - \varepsilon$ select a random action or LSTM predict action
9:          Generate another random number $\sigma$
10:         if $\sigma > \varepsilon$ then
11:             $a_t$ = Random Action Selection($s_t$)
12:         end if
13:         if $\sigma < \varepsilon$ then
14:             $a_t$ = Prediction Action Selection($s_t$)
15:         end if
16:         Otherwise select $a$ by $a^*=argmax_a Q^A(s,a)$ or $b^*=argmax_a Q^B(s,a)$
17:         Execute action $a_t$ and receive $r^t$ and $s_{t+1}$
18:         Store $(s_t, a_t, r_t, s_{t+1})$ into D
19:         Randomly sample $a$ mini-batch of experience from D
20:         Preform a gradient descent step on $Loss(\theta)$ with respect to the network parame-
    ters
21:         Choose $a$, based on $Q^A(s,\bullet)$ and $Q^B(s,\bullet)$, observe $r, s'$
22:         if UPDATE(A) then
23:             $Q^A(s,a) \leftarrow Q^A(s,a) + \rho[r + \gamma \max_{a'} Q^B(s', a^*) - Q^A(s,a)]$
24:         else if UPDATE(B) then
25:             $Q^B(s,a) \leftarrow Q^B(s,a) + \rho[r + \gamma \max_{a'} Q^A(s', b^*) - Q^B(s,a)]$
26:         end if
27:     end for
28: end for
29: Repeat
```

5. Experimental Evaluation

5.1. Experimental Setup

In this paper, we use a dataset from Google Cluster [40], which includes information about the arrival time, data size, processing time, and deadline of the tasks. These tasks include not only big tasks such as big data analysis and real-time video processing, but also small tasks such as image processing in virtual reality. Each type of task processing density, task processing time and the size of data volume are related. Therefore, we preprocess the raw data according to the characteristics of the data and make the data size compatible with the established model by normalization and denormalization. Referring to the corresponding literature [30,41], the main parameter settings are given in Table 2.

Table 2. Simulation parameters.

Parameter	Value
f_m^{device}	2.5 GHz
f_m^{EC}	41.8 GHz
B_m^{t}	10 MHz
P_m^{exe}	5 Watt
P_m^{wait}	0.2 Watt
$P_{m,n}^{\text{tran}}$	2 Watt
$P_{m,n}^{\text{exe}}$	10 Watt

5.2. Task Prediction Experiment

In a stable real edge computing scenario, the tasks generated by terminal devices are highly correlated with time and have a certain continuity and regularity. Generally, a T times history window is used to predict the task at T + 1 times. In the experiments, we set the history window to 50 and set different thresholds for the optimization target $|D_t - D_t^{\sim}|$. The experimental results are shown in Figure 6. When the threshold value is set small, the LSTM prediction model describes the historical data volume with higher accuracy and can fully explore the changing pattern of the data volume, however, it will introduce a larger prediction overhead, such as will increase the training time of the LSTM model.

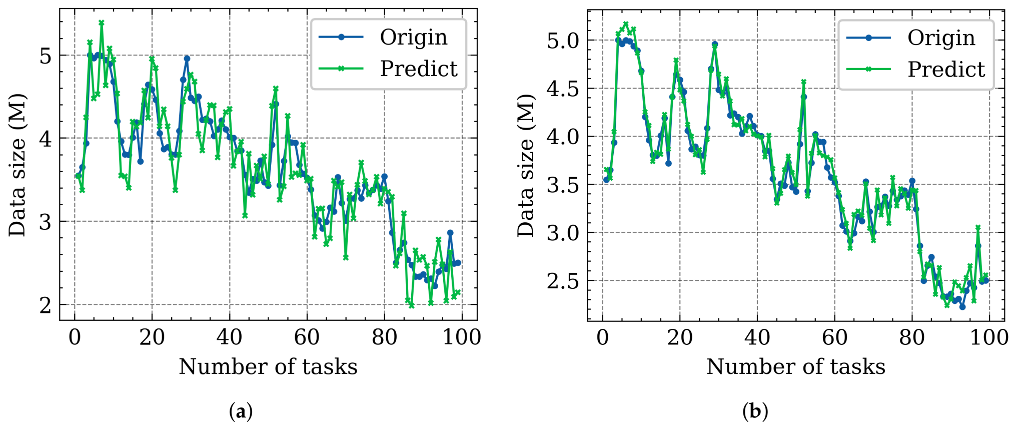


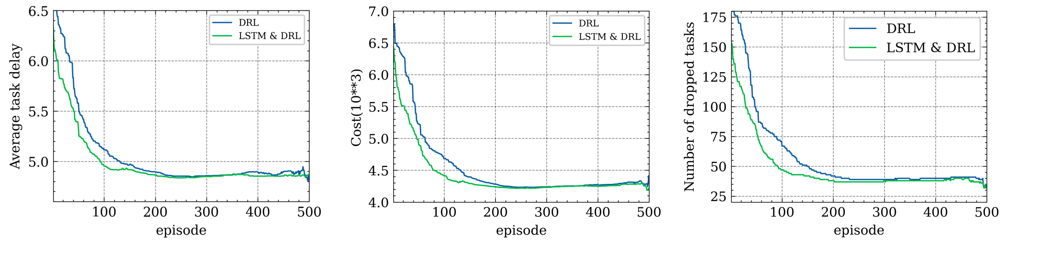
Figure 6. Effect of threshold size on LSTM prediction task features. (**a**) Threshold size = 0.5 M; (**b**) Threshold size = 0.1 M.

5.3. Training Process of LSTM & DRL

When performing training on the DRL offload decision model, it takes a longer time to explore and select the better result due to the initial random selection action of the agent. In this paper, we propose to predict the server load based on the edge server record history data. Based on the prediction results, the server predicted to be non-idle is selected with a certain probability as the offload choice for the next moment. This solution allows the agent to effectively avoid selecting servers with high loads, thus reducing task processing latency and task dropping rates. In this paper, we propose to use LSTM for load prediction and compare the impact of decisions with load prediction (LSTM & DRL) and without load prediction (DRL) on offloading performance. As can be seen in Figure 7, the traditional DRL is significantly slower than the LSTM & DRL for load prediction in the early stages of training decision making. As the number of training increases and the historical data keeps increasing, the LSTM & DRL can effectively make predictions about the load, and based on the prediction results, the agent can make full use of the best strategy that has been explored. Therefore, after certain training, the average delay, energy consumption, and the number of task throw volumes can be reduced rapidly by using LSTM for load prediction.

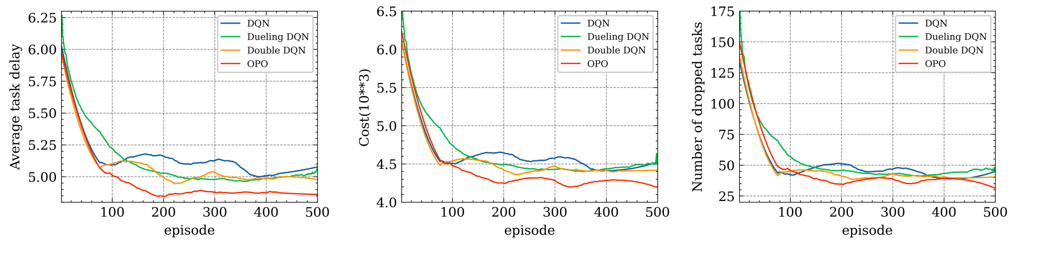
5.4. Performance Comparison

In this paper, the proposed OPO algorithm is compared with DQN, Double DQN, and Dueling DQN benchmark methods, and the following performance metrics are used: the average delay, energy consumption, and the number of task throw volume to verify the performance advantages of the proposed algorithm. The experimental results are shown in Figure 8. Five edge servers and 50 terminal devices are used for simulation experiments, and it can be obtained that the proposed algorithm significantly outperforms the other benchmark methods in all the above three performance metrics. This is because a combination of predictive and decision methods to deal with complex scenarios can make excellent use of the characteristics of the tasks and the load on the edge servers.



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

Figure 7. training process of LSTM & DRL. (**a**) Average Delay; (**b**) Tasks Costs; (**c**) Number of Dropped Tasks.

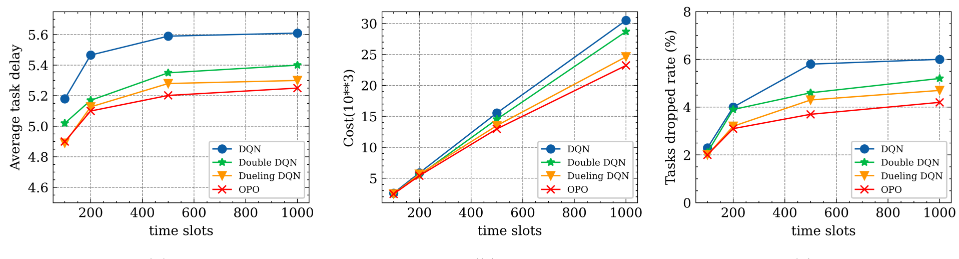


(**a**) (**b**) (**c**)

Figure 8. Comparison of different DRL algorithms. (**a**) Average delay; (**b**) tasks costs; (**c**) number of dropped tasks.

5.5. Impact of the Tasks Number

Previous studies have proved that the task arrival rate has a large impact on the system state due to the random generation of tasks. In this paper, we use data from real scenarios for data streams, so unlike traditional schemes that use task arrival rates to judge the system state, this paper uses different time slots to verify the impact of the number of tasks on the system cost, average task delay, and task discard rate. Specifically, we set the time slots in the dataset to T = 100, 200, 500, 1000, and compare the performance of DQN, double DQN, dueling DQN, and OPO under different time slots. The experimental results are shown in Figure 9. The OPO algorithm has similar trends in various performance metrics as the conventional scheme in a short period. However, as the running time of the system increases (i.e., the number of tasks increases), OPO reduces at least 6.25% of the average latency, 25.6% of the offloading cost, and 31.7% of the task drop rate compared to other algorithms in terms of cost, average latency, and task dropped rate.



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

Figure 9. Impact of tasks number on offloading decision. (**a**) Average delay; (**b**) tasks costs; (**c**) tasks dropped rate.

5.6. Impact of the Learning Rate

In this paper, we also study the convergence of the proposed algorithm at different learning rates (denoted as lr). As can be seen in Figure 10, when lr = 0.001, the algorithm is able to achieve a relatively fast convergence rate and a small convergence cost. As the learning rate decreases (i.e., below 0.0001), the convergence is slower and takes longer to reach a better value. When the learning rate is larger, the convergence cost increases and may even be higher than that of the stochastic strategy.

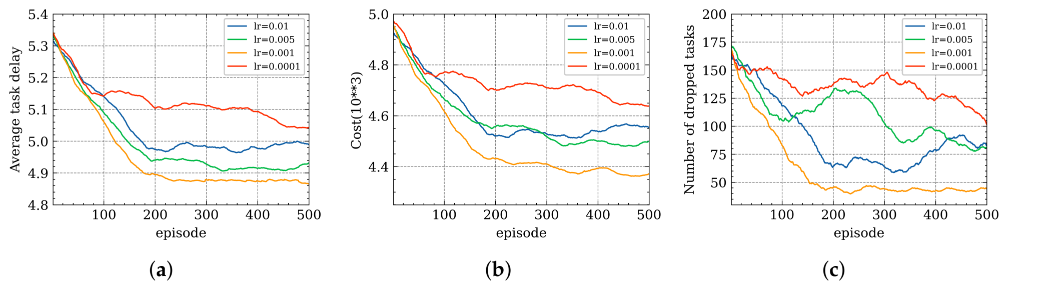


Figure 10. Impact of tasks number on offloading decision. (**a**) Average delay; (**b**) tasks costs; (**c**) number of dropped tasks.

5.7. Simulation of Real-Time Decision

The newly generated tasks have to find the best computational nodes and allocate suitable resources through the decision model. The tasks generate queuing delay and decision response delay in the cache queue from generation until the decision information is derived. Through the LSTM prediction model, the next incoming task is predicted based on the historical data of the task and enters the decision model to propose an offloading scheme for its assignment. In the experiments, we select the first 500 tasks for comparison. The experimental results are shown in Figure 11. It can be seen that at the beginning of the experiment, the tasks can choose good compute nodes because the cache queue and the edge server system are relatively idle, but as time goes by, the number of tasks increases the task stacking in the cache queue, which leads to an increase in the queuing delay of the tasks. Based on the prediction model, when the task is generated, the actual value of the task is compared with the predicted value, and when the error meets the set threshold, the task can be directly offloaded for computation according to the proposed assignment, otherwise, it waits to enter the decision model for reassignment. In the long run, task based prediction can effectively reduce the decision latency of tasks.

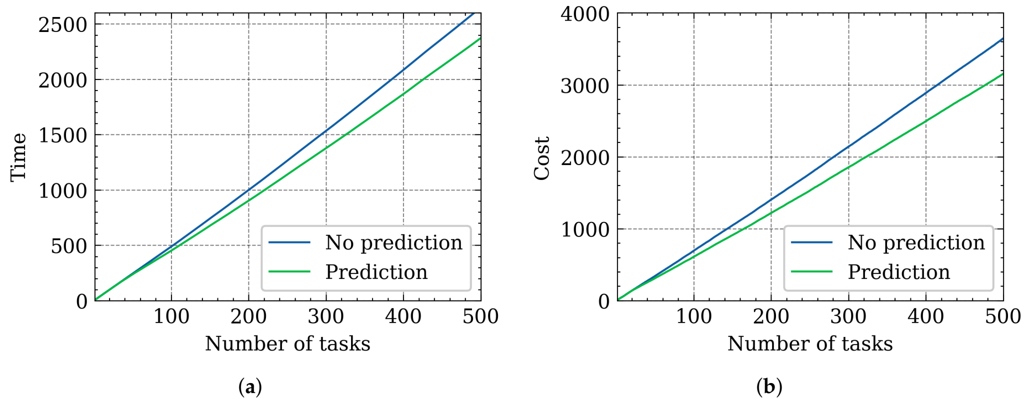


Figure 11. Diagram of real-time simulated decison results. (**a**) Total delay; (**b**) total costs.

6. Conclusions

In this paper, we study the computational offloading problem in edge computing by considering task latency, energy consumption, and discard rate. We model the offloading problem with the optimization objective of minimizing long-term cost by jointly optimizing the above metrics. The model is solved by combining the prediction method of LSTM networks and the decision method of DQN. By combining the advantages of Double DQN and Dueling DQN, an OPO algorithm based on deep reinforcement learning is proposed to improve the accuracy and stability of the model convergence. The training speed and training accuracy of the DRL model are improved by using the prediction capability of LSTM. In the actual inference process, the offloading decision delay of the task is reduced by proposing a good offloading decision for the task in advance. According to the experiment results, the OPO algorithm can provide a better offloading decision solution for the task offloading decision problem in a real IoT environment. However, the experimental results of the method in this paper are derived based on simulation techniques. In future research, we will consider migrating the method to an experimental tested, and then combining the latest algorithms and techniques to improve the performance of the OPO algorithm in real IoT applications scenarios.

Author Contributions: Data curation, Y.T. and L.Y.; Formal analysis, Y.T.; Funding acquisition, H.C. and X.Z.; Project administration, H.C.; Software, H.C.; Visualization, Y.T.; Writing—original draft, Y.T.; Writing—review & editing, H.C., L.Y. and X.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China (NSFC) grant number 62002183. This research was funded by Ningbo Natural Science Foundation grant number 2021J090. This research was funded by Ningbo Natural Science Foundation grant number 202003N4087.

Data Availability Statement: Not applicable.

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

References

1. Aslanpour, M.S.; Gill, S.S.; Toosi, A.N. Performance Evaluation Metrics for Cloud, Fog and Edge Computing: A Review, Taxonomy, Benchmarks and Standards for Future Research. *Internet Things* **2020**, *12*, 100273. [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. Abbas, N.; Zhang, Y.; Taherkordi, A.; Skeie, T. Mobile Edge Computing: A Survey. *IEEE Internet Things J.* **2018**, *5*, 450–465. [CrossRef]
4. Lin, L.; Liao, X.; Jin, H.; Li, P. Computation Offloading Toward Edge Computing. *Proc. IEEE* **2019**, *107*, 1584–1607. [CrossRef]
5. Kuang, L.; Gong, T.; OuYang, S.; Gao, H.; Deng, S. Offloading Decision Methods for Multiple Users with Structured Tasks in Edge Computing for Smart Cities. *Future Gener. Comput. Syst.* **2020**, *105*, 717–729. [CrossRef]
6. Thai, M.T.; Lin, Y.D.; Lai, Y.C.; Chien, H.T. Workload and Capacity Optimization for Cloud-Edge Computing Systems with Vertical and Horizontal Offloading. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 227–238. [CrossRef]
7. Cui, L.; Xu, C.; Yang, S.; Huang, J.Z.; Li, J.; Wang, X.; Ming, Z.; Lu, N. Joint Optimization of Energy Consumption and Latency in Mobile Edge Computing for Internet of Things. *IEEE Internet Things J.* **2018**, *6*, 4791–4803. [CrossRef]
8. Gu, Q.; Wang, G.; Liu, J.; Fan, R.; Fan, D.; Zhong, Z. Optimal Offloading with Non-Orthogonal Multiple Access in Mobile Edge Computing. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–5. [CrossRef]
9. Mukherjee, M.; Kumar, V.; Kumar, S.; Matamy, R.; Mavromoustakis, C.X.; Zhang, Q.; Shojafar, M.; Mastorakis, G. Computation Offloading Strategy in Heterogeneous Fog Computing with Energy and Delay Constraints. In Proceedings of the IEEE International Conference on Communications (ICC), Online, 7–11 June 2020; pp. 1–5. [CrossRef]
10. Wu, Y.; Shi, J.; Ni, K.; Qian, L.; Zhu, W.; Shi, Z.; Meng, L. Secrecy-Based Delay-Aware Computation Offloading via Mobile Edge Computing for Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4201–4213. [CrossRef]
11. Meng, H.; Chao, D.; Guo, Q. Deep Reinforcement Learning Based Task Offloading Algorithm for Mobile-Edge Computing Systems. In Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence, Chegndu, China, 12–15 April 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 90–94. [CrossRef]
12. Huang, L.; Bi, S.; Zhang, Y.J.A. Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2020**, *19*, 2581–2593. [CrossRef]
13. Yan, P.; Choudhury, S. Optimizing Mobile Edge Computing Multi-Level Task Offloading via Deep Reinforcement Learning. In Proceedings of the IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; IEEE: New York, NY, USA, 2020; pp. 1–7. [CrossRef]
14. Huang, L.; Feng, X.; Zhang, C.; Qian, L.; Wu, Y. Deep Reinforcement Learning-based Joint Task Offloading and Bandwidth Allocation for Multi-user Mobile Edge Computing. *Digit. Commun. Netw.* **2019**, *5*, 10–17. [CrossRef]
15. Kumar, R.; Kumar, P.; Kumar, Y. Time Series Data Prediction using IoT and Machine Learning Technique. *Procedia Comput. Sci.* **2020**, *167*, 373–381. [CrossRef]

16. Abdellah, A.R.; Mahmood, O.A.K.; Paramonov, A.; Koucheryavy, A. IoT Traffic Prediction Using Multi-step Ahead Prediction with Neural Network. In Proceedings of the 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Dublin, Ireland, 28–30 October 2019; pp. 1–4. [CrossRef]
17. Gao, J.; Wang, H.; Shen, H. Machine Learning Based Workload Prediction in Cloud Computing. In Proceedings of the 2020 29th International Conference on Computer Communications and Networks (ICCCN), Online, 3–6 August 2020; pp. 1–9. [CrossRef]
18. Sonmez, C.; Tunca, C.; Ozgovde, A.; Ersoy, C. Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 2239–2251. [CrossRef]
19. 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.* **2020**, *7*, 1678–1689. [CrossRef]
20. Guo, H.; Liu, J. Collaborative Computation Offloading for Multiaccess Edge Computing Over Fiber–Wireless Networks. *IEEE Trans. Veh. Technol.* **2018**, *67*, 4514–4526. [CrossRef]
21. Ali, R.; Zikria, Y.B.; Garg, S.; Bashir, A.K.; Obaidat, M.S.; Kim, H.S. A Federated Reinforcement Learning Framework for Incumbent Technologies in Beyond 5G Networks. *IEEE Netw.* **2021**, *35*, 152–159. [CrossRef]
22. Ali, R.; Ashraf, I.; Bashir, A.K.; Zikria, Y.B. Reinforcement-Learning-Enabled Massive Internet of Things for 6G Wireless Communications. *IEEE Commun. Stand. Mag.* **2021**, *5*, 126–131. [CrossRef]
23. Zhao, T.; Zhou, S.; Song, L.; Jiang, Z.; Guo, X.; Niu, Z. Energy-optimal and Delay-bounded Computation Offloading in Mobile Edge Computing with Heterogeneous Clouds. *China Commun.* **2020**, *17*, 191–210. [CrossRef]
24. Vu, T.T.; Huynh, N.V.; Hoang, D.T.; Nguyen, D.N.; Dutkiewicz, E. Offloading Energy Efficiency with Delay Constraint for Cooperative Mobile Edge Computing Networks. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6. [CrossRef]
25. Yuan, H.; Zhou, M. Profit-Maximized Collaborative Computation Offloading and Resource Allocation in Distributed Cloud and Edge Computing Systems. *IEEE Trans. Autom. Sci. Eng.* **2021**, *18*, 1277–1287. [CrossRef]
26. Alqerm, I.; Pan, J. DeepEdge: A New QoE-Based Resource Allocation Framework Using Deep Reinforcement Learning for Future Heterogeneous Edge-IoT Applications. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 3942–3954. [CrossRef]
27. Amin, F.; Ahmad, A.; Sang Choi, G. Towards Trust and Friendliness Approaches in the Social Internet of Things. *Appl. Sci.* **2019**, *9*, 166. [CrossRef]
28. Chen, W.; Qiu, X.; Cai, T.; Dai, H.N.; Zheng, Z.; Zhang, Y. Deep Reinforcement Learning for Internet of Things: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1659–1692. [CrossRef]
29. Huang, L.; Feng, X.; Feng, A.; Huang, Y.; Qian, L.P. Distributed Deep Learning-based Offloading for Mobile Edge Computing Networks. *Mob. Netw. Appl.* **2018**, 1–8. [CrossRef]
30. Tang, M.; Wong, V.W. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Trans. Mob. Comput.* **2020**, 1. [CrossRef]
31. Jang, I.; Kim, H.; Lee, D.; Son, Y.S.; Kim, S. Knowledge Transfer for On-Device Deep Reinforcement Learning in Resource Constrained Edge Computing Systems. *IEEE Access* **2020**, *8*, 146588–146597. [CrossRef]
32. Gong, Y.; Wang, J.; Nie, T. Deep Reinforcement Learning Aided Computation Offloading and Resource Allocation for IoT. In Proceedings of the 2020 IEEE Computing, Communications and IoT Applications (ComComAp), Beijing, China, 5–8 November 2020; pp. 1–6. [CrossRef]
33. Chen, X.; Zhang, H.; Wu, C.; Mao, S.; Ji, Y.; Bennis, M. Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning. *IEEE Internet Things J.* **2019**, *6*, 4005–4018. [CrossRef]
34. Song, S.; Fang, Z.; Zhang, Z.; Chen, C.L.; Sun, H. Semi-Online Computational Offloading by Dueling Deep-Q Network for User Behavior Prediction. *IEEE Access* **2020**, *8*, 118192–118204. [CrossRef]
35. Zou, J.; Hao, T.; Yu, C.; Jin, H. A3C-DO: A Regional Resource Scheduling Framework Based on Deep Reinforcement Learning in Edge Scenario. *IEEE Trans. Comput.* **2021**, *70*, 228–239. [CrossRef]
36. Liu, L.; Feng, J.; Pei, Q.; Chen, C.; Ming, Y.; Shang, B.; Dong, M. Blockchain-Enabled Secure Data Sharing Scheme in Mobile-Edge Computing: An Asynchronous Advantage Actor–Critic Learning Approach. *IEEE Internet Things J.* **2021**, *8*, 2342–2353. [CrossRef]
37. Fu, F.; Kang, Y.; Zhang, Z.; Yu, F.R.; Wu, T. Soft Actor–Critic DRL for Live Transcoding and Streaming in Vehicular Fog-Computing-Enabled IoV. *IEEE Internet Things J.* **2021**, *8*, 1308–1321. [CrossRef]
38. Chen, J.; Chen, S.; Wang, Q.; Cao, B.; Feng, G.; Hu, J. iRAF: A Deep Reinforcement Learning Approach for Collaborative Mobile Edge Computing IoT Networks. *IEEE Internet Things J.* **2019**, *6*, 7011–7024. [CrossRef]
39. Chen, J.; Chen, S.; Luo, S.; Wang, Q.; Cao, B.; Li, X. An Intelligent Task Offloading Algorithm (iTOA) for UAV Edge Computing Network. *Digit. Commun. Netw.* **2020**, *6*, 433–443. [CrossRef]
40. Yuan, H.; Tang, G.; Li, X.; Guo, D.; Luo, L.; Luo, X. Online Dispatching and Fair Scheduling of Edge Computing Tasks: A Learning-Based Approach. *IEEE Internet Things J.* **2021**, *8*, 14985–14998. [CrossRef]
41. Chen, J.; Xing, H.; Xiao, Z.; Xu, L.; Tao, T. A DRL Agent for Jointly Optimizing Computation Offloading and Resource Allocation in MEC. *IEEE Internet Things J.* **2021**, *8*, 17508–17524. [CrossRef]