applied sciences

MDPI

Article

Dynamic Computation Offloading with Deep Reinforcement Learning in Edge Network

Yang Bai [1], Xiaocui Li [1], Xinfan Wu [1] and Zhangbing Zhou [1,2,*]

[1] School of Information Engineering, China University of Geosciences (Beijing), Beijing 100083, China
[2] Computer Science Department, TELECOM SudParis, 91000 Evry, France
* Correspondence: zbzhou@cugb.edu.cn

Abstract: With the booming proliferation of user requests in the Internet of Things (IoT) network, Edge Computing (EC) is emerging as a promising paradigm for the provision of flexible and reliable services. Considering the resource constraints of IoT devices, for some delay-aware user requests, a heavy-workload IoT device may not respond on time. EC has sparked a popular wave of offloading user requests to edge servers at the edge of the network. The orchestration of user-requested offloading schemes creates a remarkable challenge regarding the delay in user requests and the energy consumption of IoT devices in edge networks. To solve this challenge, we propose a dynamic computation offloading strategy consisting of the following: (i) we propose the concept of intermediate nodes, which can minimize the delay in user requests and the energy consumption of the current tasks handled by IoT devices by dynamically combining task-offloading and service migration strategies; (ii) based on the workload of the current network, the intermediate node selection problem is modeled as a multi-dimensional Markov Decision Process (MDP) space, and a deep reinforcement learning algorithm is implemented to reduce the large MDP space and make a fast decision. Experimental results show that this strategy is superior to the existing baseline methods to reduce delays in user requests and the energy consumption of IoT devices.

Keywords: computation offloading; service migration; deep reinforcement learning

check for updates

Citation: Bai, Y.; Li, X.; Wu, X.; Zhou, Z. Dynamic Computation Offloading with Deep Reinforcement Learning in Edge Network. *Appl. Sci.* **2023**, *13*, 2010. https://doi.org/10.3390/app13032010

Academic Editor: Gianluigi Ferrari

Received: 8 January 2023
Revised: 29 January 2023
Accepted: 31 January 2023
Published: 3 February 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 booming development of the Internet of Things (IoT), millions of sensors and devices are being deployed at the network's edge [1], generating vast amounts of data. Due to the limited storage and computing resources, these IoT devices face significant challenges [2] in storing and processing the data. An effective solution is to offload complex tasks to the remote cloud. By making use of the rich computing resources of the cloud-computing platform, the task response time and energy consumption of IoT devices can be reduced. However, for some delay-sensitive applications, such as real-time video streaming [3] and real-time face recognition [4], cloud computing may not be able to respond in time, due to the high communication overhead and limited network bandwidth. To resolve the above contradictions, CISCO proposed an Edge Computing (EC) paradigm [5]. In the EC paradigm, spatially distributed resource servers, called edge servers, form a large-scale computing network [6]. Using these edge servers, data storage and processing is carried out outside the data generation device, reducing the storage and data processing overhead. Thus, the EC paradigm efficiently increases the network capacity [7] and reduces the energy consumption of IoT devices. Moreover, geographically, due to the edge server's proximity to the users, the EC paradigm significantly improves the Quality of Service (QoS). However, due to the proliferation of computation-intensive tasks and edge servers' limited resources, the edge server cannot process all task requests in a timely manner. Thus, computation offloading was proposed to relieve the heavy workload of the edge servers.

Many researches are focused on reducing network energy consumption and service response delays by computation offloading. For example, an alternating minimizing

algorithm is proposed to achieve energy-optimal computation offloading [8]. To minimize the overall offloading time, the authors designed a delay-dependent, priority-aware, task-offloading strategy in [9], which assigns priority to each task according to its deadline. These papers concentrate on partial offloading strategies or the joint optimization of offloading decisions and resource allocation [10–14]. However, most of the related research only focuses on task offloading to tackle the problems of service response latency and the limited energy consumption of IoT devices, ignoring the fact that the users' service requests are highly dynamic in practice. Edge servers may not configure the services required by the task, which will constrain the task offloading strategy. Service migration from the cloud to the network's edge is introduced in [15] to resolve the conflict between the diversity of service requests and the limited number of services on the edge server. However, the communication time and energy cost regarding communication between cloud servers and edge servers are high. As a result, the frequent migration of services from the cloud to edge servers will lead to an increased user response latency and increased energy consumption by IoT devices. One possible solution to this problem is to migrate services between edge servers, bringing available services closer to the user devices, and thereby reducing the service migration's extra cost [16]. To achieve the best balance between QoS and migration costs, and bring services closer to users, when and where [17] to migrate services must be determined. These works aim to achieve low-latency service migration and seamless computation in a dynamic traffic environment in [18–21]. An autonomous bandwidth allocation approach is proposed to suppor low-latency service migration [22]. The authors used an algorithm based on decision theory to find an optimal service migration path [23]. Most of the existing works concentrated on finding an optimal algorithm to lower the delay in task offloading or service migration. However, the decision time of the algorithm itself is ignored. For delay-sensitive applications and the unstable network environments, the complexity and robustness of decision-making algorithms are key to fast decision-making [24]. In general, the problem of task-offloading and service migration can be expressed as a Markov Decision Process (MDP) [25]. The use of reinforcement learning algorithms to find the optimal MDP strategy [26] is a great choice. Among the various reinforcement learning algorithms, the Q-learning algorithm has the advantage of allowing for fast decision-making [27]. Nevertheless, for large sets of states and actions, the traditional Q-learning algorithm cannot solve large-scale MDP problems. Deep learning can use Deep Neural Networks (DNNs) to learn very complex functions and deal with high-dimensional data samples. By combining q-learning with deep learning, a Deep Q-leaning Network (DQN) reinforcement learning model is proposed, which inspired our work [28]. Collaboration among edge servers, such as task offloading and service migration, essentially improves the QoS and reduces energy consumption in the EC paradigm. Most of the related works consider the two collaboration strategies separately, and do not efficiently combine the two strategies.

To meet this challenge, this paper proposes an effective computational offloading strategy, which minimizes the task-processing cost by dynamically combining task-offloading and service migration strategies. Our main contributions can be summarized as follows:

- We propose a novel dynamic computation offloading strategy, where the computation offloading and service migration are combined to select intermediate nodes with sufficient resources. This strategy can minimize the cost of task processing user requests.
- We model this kind of optimal intermediate node selection problem as an MDP and implement deep reinforcement learning algorithms to reduce the large MDP space and achieve fast decision-making.
- A large number of simulations are carried out to verify the effectiveness of this strategy in reducing both the delay and the energy consumption.

The rest of this paper is organized as follows. Section 2 reviews and discusses relevant works. Section 3 presents the three-tier system model, delay model, energy consumption model and problem formulation. Section 4 presents a detailed description of our dynamic

computation strategy and deep reinforcement learning-based algorithm. Section 5 conducts extensive simulations to validate this algorithm's efficiency. Section 6 concludes this work.

2. Related Work

With the growth of computation-intensive tasks and the diversification of service requests, computation offloading and service migration in EC have attracted increased attention. Many researchers are committed to designing effective algorithms to improve QoS and reduce costs. The relevant algorithms and models are discussed in the following.

2.1. Task Offloading in Edge Computing (EC)

The decision-making factor in task offloading plays a key role in edge networks. Plenty of works have designed efficient algorithms or models. The authors of [8] designed an Energy-optimal Dynamic Computation Offloading (EDCO) scheme by jointly optimizing various parameters. The research has proposed a novel Delay-dependent Priority-aware Task-Offloading (DPTO) strategy for task scheduling [9]. A socially aware dynamic computation offloading scheme was proposed in [10]. Some researchers considered each fog node's time–energy-efficiency and priority to propose a task-offloading scheme called Fair Task-Offloading (FTO) [11]. Some studies have focused on the partial offloading approach to achieve a trade-off between the energy consumption of IoT devices [12] and task-processing delays [13,14]. The authors introduced a new dynamic edge-computing model and designed an online primal-dual algorithm to offload the task when it arrives [29]. In reference [30], a new optimization model to maximize the expected unloading rate of agents is proposed, using the game-theory method. A novel task-offloading method based on meta-reinforcement learning is proposed to improve the generalization ability of the offloading algorithm [31]. Based on a fog resource, a Fog Resource-aware Adaptive Task-Offloading (FRATO) framework is proposed to flexibly select the optimal offloading policy [32]. The authors proposed a computation offloading and resource allocation strategy named Deep Deterministic Policy Gradient (DDPG), based on the deep reinforcement learning presented in [33].

However, these works did not consider the service that was requested by users, and may not be configured on the execution node. In this work, we introduce a dynamic computation offloading strategy, which takes the intermediate nodes with rich computing resources as the execution nodes in the task. If the execution node does not configure the corresponding service, the service migration process will occur in parallel. The deep reinforcement learning algorithm is used to help the optimal intermediate node make a quick decision.

2.2. Service Migration in Edge Computing (EC)

The limited services of edge servers and the users' mobility bring more challenges to the EC paradigm. Service migration between edge servies has attracted attention as a means of achieving low-latency and seamless computation. A service migration strategy based on user mobility was proposed [34]. As the user device moves, the service migrates to another edge server. The literature [35] developed an optimal iterative relaxation-and-rounding-based solution approach to minimize the migration cost. Considering the high mobility of vehicles, the authors transform the service migration problem into an uncertain decision optimization problem and designed a Latency-aware Service Migration method with Decision theory (LSMD) [23]. However, it is difficult to achieve fast algorithm convergence due to the traditional heuristic algorithm's high time complexity. The decision process of service migration is modeled as a one-dimensional MDP [36]. Therefore, it is appropriate to use the reinforcement learning algorithm for service migration decisions. Reinforcement learning is emerging as a promising approach to optimize service migration strategies in academics. The migration strategy was optimized based on reinforcement learning, with the service migration delay as the objective function [37]. In addition, some researchers proposed a novel service migration algorithm and architecture to support

mobility tasks based on reinforcement learning, which can efficiently reduce the extra delay and energy costs of the migration process [38]. The literature [39] modeled the service migration problem as a complex optimization and implemented deep reinforcement learning to approximate the optimal policy.

Our work also adopted deep reinforcement learning to achieve quick decision-making, and we dynamically combined the service migration with the task-offloading. Decisions regarding when and where to migrate the task's corresponding service were related to our dynamic computation offloading strategy.

3. Modeling And Problem Formulation

3.1. System Model

As shown in Figure 1, the system's architecture has three layers: the user layer, edge server layer, and the cloud layer [40]. The user layer includes IoT devices. IoT devices collect the data needed for various services and transmit them to the nearest edge server through wireless communication for further processing. These IoT devices can send service requests and broadcast them to all edge servers. These service requests will be received and processed by the nearest edge server. They can also be offloaded to other idle edge servers via access networks [41]. In the edge server layer, a service can respond to requests from multiple IoT devices. Each task generated by devices is served by a corresponding service, which shares its physical resources with other services on the same edge server [42]. Edge servers not only provide computational resources for task processing, but also determine the computation offloading strategy [43]. The cloud layer is far from IoT devices and edge servers. The cloud is mainly responsible for data replication and load-balancing, not computing.

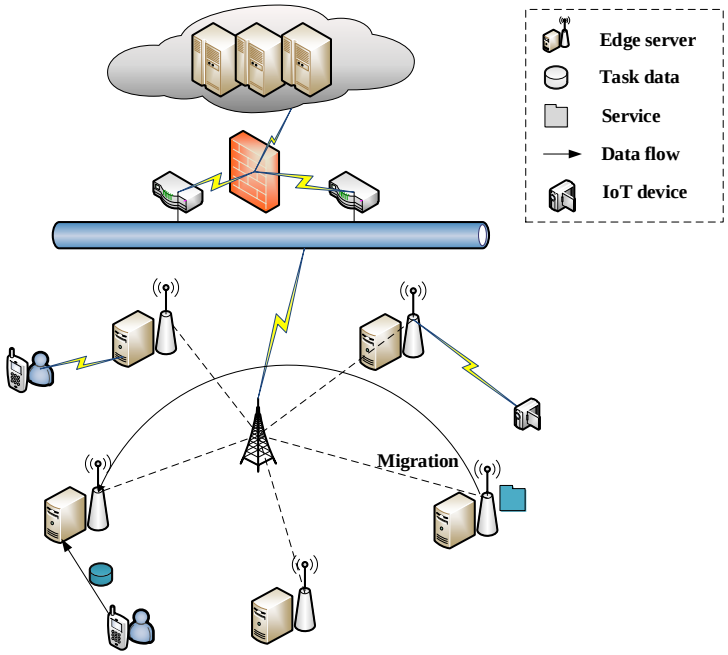


Figure 1. System architecture.

Based on the system model described above, we present some of the notations used in this paper. $F = \{F_1, F_2, \ldots, F_n\}$, representing the set of edge servers described by the following attributes: the unique identifier of each server $F_i.id$, the geographic location $F_i.loc = (L_x, L_y)$, the processing capacity of edge server $F_i.cap$, the list of services currently hosted by the edge server $F_i.hostLst$, and the list of tasks currently being processed by the edge server $F_i.pt$. The main purpose of this paper is to find a better way to reduce the delays in task processing and the energy consumption of the edge networks, instead of studying the edge server heterogeneity. Therefore, we assume that all edge servers have the

same physical resources, which include CPU processing capacity, bandwidth, memory, and storage. In the realistic setting, services are highly diverse. Without loss of generality, we assume that the edge server layer includes m types of services for simplicity, labeled as a set $H = \{H_1, H_2, \ldots, H_m\}$, which is described as having the following attributes: the unique identifier of the service $H_i.id$; the service capacity $H_i.bytes$. Dividing the time period T evenly into time slots, the time slot set can be represented by $T = \{T_1, T_2, \ldots, T_t\}$. In each time slot, the IoT devices at the user layer are assumed to issue k task requests, labelled as $C = \{C_1, C_2, \ldots, C_k\}$, which can be described by the following attributes: the unique identifier of the task $C_i.id$, the geographic location $C_i.loc = (L_x, L_y)$, the service type of the task $C_i.H_j$, the capacity of the task data $C_i.bytes$ and the number of CPU cycles required for task processing $C_i.cir$.

3.2. Delay Model

In this paper, the delay model includes the task-offloading delay, the task queuing delay, the task processing delay and the service migration delay. Detailed definitions and formulas for these involved delay models are as follows.

3.2.1. Task-Offloading Delay

The task-offloading delay is defined as the time taken for data transmission between the local edge server and target edge server. The time needed for task offloading is related to the data capacity and the transmission speed. Therefore, the data transmission delay from user C_i to the edge server F_j at time slot T_t can be expressed as follows:

$$d_{to}^{i,j,t} = \frac{C_i.bytes}{V_{tr}^{i,j,t}} \tag{1}$$

where $C_i.bytes$ is the capacity of the task data; $V_{tr}^{i,j,t}$ is the data transmission speed and can be defined as:

$$V_{tr}^{i,j,t} = Blog_2(1 + \frac{P_{tx}^i G^{i,j,t}}{N_{noise}}) \tag{2}$$

where B represents the channel bandwidth; P_{tx}^i represents the transmission power of user device C_i; $G^{i,j,t}$ represents the channel gain between the user device C_i and edge server F_j; N_{noise} is the background noise.

3.2.2. Task Processing Delay

The task request needs to process on the edge server, which requires computing resources. Therefore, the processing delay is mainly related to the edge server's computation speed. In the scenario in this article, the physical resources of all edge servers are the same as those for simplicity. Therefore, the processing delay in the task request can be formulated as follows:

$$d_{pr}^{i,j,t} = \frac{C_i.cir}{F_j.cap} \tag{3}$$

where $C_i.cir$ represents the number of CPU cycles required for the task and $F_j.cap$ is the CPU frequency that denotes the edge server's processing capacity.

3.2.3. Task Queuing Delay

If the edge serve is idle when the task arrives, it will immediately start processing the task. Otherwise, the task will be added to the queue. After all the queue tasks are processed, the edge server loads the corresponding service to process the current task. Therefore, the queuing time is the wait time from when the task arrives at the node to the moment it starts

to be processed. If we assume that K tasks precede the current task, then, from Equation (3), we can obtain:

$$d_{tq}^{i,j,t} = \sum_{k=1}^{K} \frac{C_k.cir}{F_j.cap} \tag{4}$$

3.2.4. Service Migration Delay

The delay in service migration is similar to the delay in task offloading. This is mainly related to the service capacity and the transmission speed. The migration decision time is negligible compared to the transmission delay [44]. The migration delay can be expressed as:

$$d_{cm}^{i,t} = \frac{H_i.bytes}{V_{tr}^{j,j',t}} + d_r \tag{5}$$

where $H_i.bytes$ is the service capacity, $V_{tr}^{j,j',t}$ is the transmission speed between original edge server j to the migration target edge server j' , and d_r is the service reconfiguration time.

3.3. Energy Consumption Model

Various energy consumption models were proposed for the IoT scenarios. The First-Order Ratio Model (FORM) [45] is more widely used. In this paper, the model is adopted and simplified to make it more suitable for the application scenario in this problem. Our energy consumption model includes task transmission consumption, service migration consumption and task processing consumption. The detailed definitions and formulas of these energy consumption rates are as follows.

3.3.1. Data Transmission Energy Consumption

The data transmission energy consumption consists of two parts: the task-offloading energy consumption, and the migration energy consumption. These are both, essentially, the energy consumed by data transmission, so their formula definitions are the same. Based on FORM, the task transmission energy consumption $e_{to}^{i,j,t}$ and the migration energy consumption $e_{cm}^{j,j',t}$ can be expressed as follows:

$$e_{to}^{i,j,t} = e_{Tx}(k,d) + e_{Rx}(k) \quad i \in C, j \in F \tag{6}$$

$$e_{cm}^{j,j',t} = e_{Tx}(k,d) + e_{Rx}(k) \quad j \in F, j \in F \tag{7}$$

$$e_{Tx}(k,d) = e_{elec} \times k + \epsilon_{amp} \times k \times d^n \tag{8}$$

$$e_{Rx}(k) = e_{elec} \times k \tag{9}$$

where $e_{Tx}(k,d)$ is the energy consumed to increase data capacity k to an edge server at distance d; $e_{Rx}(k)$ represents the energy that is required to receive k bits of data. The environment strongly influences the value of the propagation fading index n. If there is no building obstruction, the value of n can be set to 2 when transmitting data between user devices and edge servers. Otherwise, it can be 3, 4, or 5, depending on the specific environment. e_{elec} and ϵ_{amp} represent the energy consumption constant of the transmission and receiver electronics and the energy consumption constant of the transmit amplifier, respectively.

3.3.2. Processing Energy Consumption

The energy consumption of an edge server is determined by a combination of storage, memory and CPU usage. Since the CPU dominates these factors, this is the main focus of our work. A model based on DVFS technology [46] is described as follows. An edge server handles K computational tasks. The k-th task needs $C_i.cir$ CPU cycles, and the CPU cycle

frequency is $F_i.cap$. The total energy consumed by the CPU on the edge server, denoted by e_{pr}, can be expressed as:

$$e_{pr} = \sum_{k=1}^{K} kC_i.cirF_i.cap^2 \tag{10}$$

3.4. Problem Formulation

Our algorithm aims to reduce the delays in user requests and the energy consumption of IoT devices in the edge networks. Based on the delay model and energy consumption model mentioned above, from user i at time slot t, we can present the total time delay equation and energy consumption equation as:

$$d_{total}^{i,t} = d_{to}^{i,t} + d_{pr}^{i,t} + d_{tq}^{i,t} + d_{cm}^{i,t} \tag{11}$$

$$e_{total}^{i,t} = e_{to}^{i,t} + e_{cm}^{i,t} + e_{pr}^{i,t} \tag{12}$$

Generally, the network's delay in user requests and the energy consumption of IoT devices can be defined as a weighted sum of all tasks' energy consumption and delays. Hence, the optimizing variables $\mathcal{C}$ can be denoted as:

$$\mathcal{C} = \omega_1 \sum_{i=1}^{n} d_{total}^{i,t} + \omega_2 \sum_{i=1}^{n} e_{total}^{i,t} \quad \omega_1 \in (0,1], \omega_2 \in (0,1] \tag{13}$$

where ω_1 and ω_2 denote the weights of d_{total} and e_{total} in the objective function, respectively. By adjusting the values of ω_1 and ω_2, the degree of influence that delays in user requests and the energy consumption of IoT devices have on the total cost can be adjusted accordingly. The optimization objective of this paper is to minimize the value of $\mathcal{C}$.

4. Algorithm

To minimize the total cost $\mathcal{C}$ given in Equation (13), we proposed a novel Dynamic Computation Offloading Strategy (DCOS) based on deep reinforcement learning. In Section 4.1, we present a detailed description of this strategy. The algorithm formulation of this strategy, based on deep reinforcement learning, is given in Section 4.2.

4.1. Dynamic Computation Offloading Strategy

Task offloading and service migration are promising strategies to meet the stringent requirements regarding the delay in user requests and the energy consumption of IoT devices in edge networks. There are two main causes of the task offloading strategy:

- The workload on the local edge server is heavy, which significantly increases the task processing delay, leading to QoS reductions.
- The corresponding service for the task request is not configured on the local edge server.

In both cases, tasks need to be offloaded to other edge servers and configured with the related services.

In the service migration strategy, if the local edge server does not have the relevant service, it is necessary to migrate the service from the cloud or other edge servers configured by the service to the local edge server to process the task.

As shown in Figure 2, the processing of a single task can be split into three parts: the transmission process, the queuing process, and the computing process. The transmission processes in the two abovementioned strategies are the offloading and migration process, respectively. We can also see that the tasks can only be processed at the local edge server or the other edge servers that are configured with the related service, whether offloading or migration tasks. These two schemes limit the processing location of the tasks to certain network servers and do not fully exploit the potential of collaboration between edge servers.

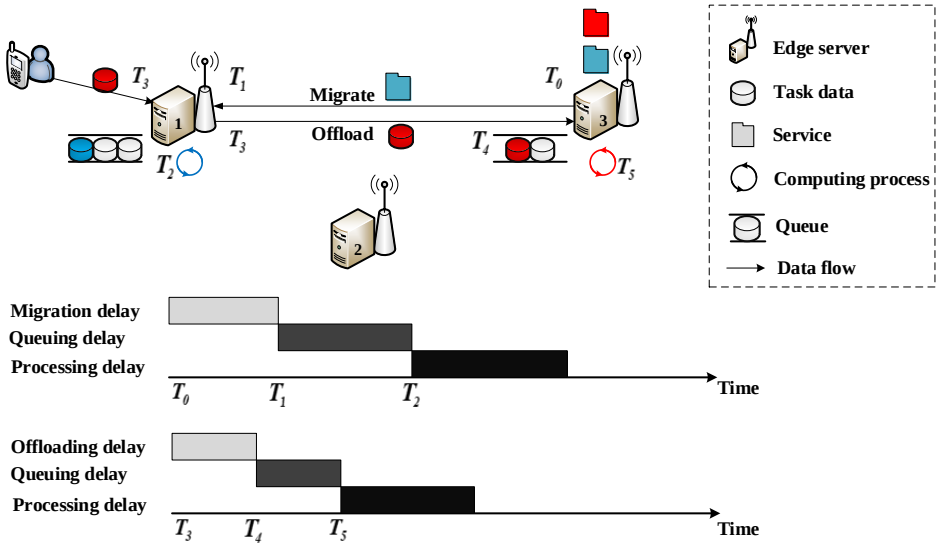


Figure 2. Task processing for migration and offloading strategies.

In our strategy, the task-processing location is extended to any server in the edge network. More specifically, when an edge server receives a task request, if the local workload is heavy or the corresponding service is not configured, the task will be offloaded to another intermediate node with sufficient computing resources. It is worth noting that the intermediate node may not be configured with the corresponding service. Therefore, it is necessary to migrate the related service from the neighboring nodes to the intermediate node. It is worth mentioning that this migration process is executed in parallel with the offloading process.

As Figure 3 shows, at time slot T_3, F_1 receives a task request from a user. The heavy workload of F_1 means that the task has a long queuing time, and no related service is configured at F_1. Therefore, the task is offloaded to the intermediate node F_2, which has sufficient computing resources. Meanwhile, the service corresponding to the task migrates from F_3 to F_2. It can be observed that the queuing time is eliminated by its execution on F_2 because it is idle. Moreover, since the location of F_2 is between F_1 and F_3, the transmission distances for both offloading and migration are correspondingly reduced compared to the offloading and migration process between F_1 and F_3 in Figure 2. Thus, the transmission delay is reduced for both. Furthermore, since both transmission processes are executed in parallel, there is some overlap between the offloading delay and migration delay.

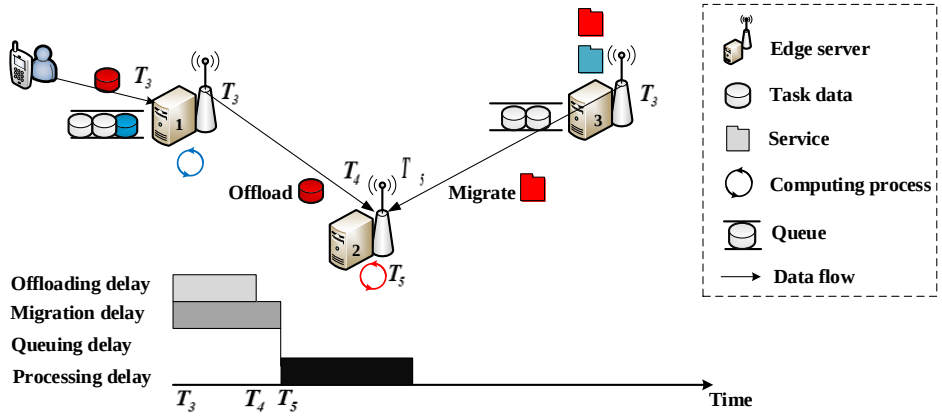


Figure 3. Task processing for dynamic computation offloading strategy.

It is worth noting that the intermediate node mentioned above is not a necessary geographic intermediate. Our strategy is based on edge servers' computing resources; even

if a node is far from the local node, we can still target it as the processing node for the task because it has sufficient computing resources, and can more efficiently process the task.

Based on the above description, we can classify our strategy's transmission process in the following three cases:

- Service migration process: the total transmission delay is the migration delay d_{cm}.
- Task offloading process: the total transmission delay is the offloading delay d_{to}.
- Parallel service migration process and task offloading process: the total transmission delay is $max(d_{cm}, d_{to})$.

Therefore, we can rewrite the Equation (11) as follows:

$$d_{total}^{i,t} = max(d_{to}^{i,t}, d_{cm}^{i,t}) + d_{pr}^{i,t} + d_{tq}^{i,t} \tag{14}$$

From this formula, a combination of task-offloading with service migration can effectively reduce the transmission delay. In addition, because the strategy is based on computing resources, processing tasks on the intermediate node can reduce the processing delay in these tasks. Overall, this dynamic computation-offloading strategy can significantly reduce the total delay in task processing.

As one aim is to reduce the cost $\mathcal{C}$ of the network, selecting the optimal intermediate node is a key problem in this strategy. Moreover, the selection of optimal intermediate nodes is a sequential decision-making process with no memory. Therefore, it is appropriate to use reinforcement learning to solve this problem. In the next section, we describe the dynamic computation-offloading algorithm based on deep reinforcement learning in.

4.2. DQN-Based Computation Offloading Algorithm

4.2.1. Reinforcement Learning Settings

For each time slot T_i in the reinforcement learning algorithm, the agent receives the system state s_i and computes the reward r_{i-1} for the last time slot. Then, the agent selects the action a_i according to a predefined policy. After implementing this action, the system moves to the new state, s_{i+1}, and into the next time slot. Similarly, the agent computes the reward r_i and selects the new action a_{i+1} according to s_{i+1}.

State: In our work, the system state is formulated by the edge servers' operation status F_{rt}, edge servers' deployed service list $F_{hostLst}$, the nearest edge server F_{near} to the current task, and the service type $H_{service}$ which the current task requires. Therefore, we use the state s_i during T_i as

$$s_i = \{F_{rt}, F_{hostLst}, F_{near}, H_{service}\} \in \mathbb{S} \tag{15}$$

The edge servers' operation status F_{rt} refers to the time cost t before the current task, which is processed on edge server $F_i(i \in n)$. Since our strategy is based on computing resources, the time cost t is related to the decision node. If the decision node is not the nearest edge server F_{near}, then the task needs to be offloaded to the decision node. The related service needs to migrate from a neighboring server if the decision node does not have the corresponding service. It is noteworthy that we use one-hot coding to formulate the F_{near} and $H_{service}$. $\mathbb{S}$ denotes the space of the system states.

Action: The selected action during T_i is $a_i = \{F_{decision}\} \in \mathbb{A}$, where $F_{decision}$ is the task's execution node, as determined by our algorithm, and $\mathbb{A}$ is the set of all possible edge servers. In DQN, DNN outputs Q-values of all actions in action space $\mathbb{A}$ based on the input state s_i, and the execution node is the output action. The action selection is based on the ϵ-greedy algorithm, a threshold ϵ is set in advance, and a random number ϕ is generated each time. If $\epsilon > \phi$, the output action is the best action corresponding to the optimal Q-value in all outputs. Otherwise, the action is randomly selected to remove local optimization.

Reward: Since our algorithm focuses on minimizing the total cost $\mathcal{C}$, if the system completes the task and there is a reduction in cost, the reward will be positive. If the cost of

the task is greater than the cost of performing the task locally, the action will face a negative reward. Therefore, the reward during T_i can be expressed as:

$$r_i = M - \frac{\mathcal{C}_{local}^{T_i}}{\mathcal{C}^{T_i}(s_i, a_i)} \tag{16}$$

where M is a constant value; $\mathcal{C}_{local}^{T_i}$ is the local execution cost at the time T_i; $\mathcal{C}^{T_i}(s_i, a_i)$ is the total cost, obtained by choosing the action a_i at the state s_i.

In the Q-learning algorithm [27], the large scale of $\mathbb{S}$ and $\mathbb{A}$ lead to the large scale of the Q-table. However, because the dimension of the state space $\mathbb{S}$ is very large, the Q-table cannot efficiently store these states. To tackle this issue, we adopted the DQN algorithm [28] in our work. DQN combines q-learning and deep learning, and uses a Q-network composed of a DNN with weights θ instead of Q-table, to effectively store Q-value information. The detailed algorithm steps of DQN are shown in Algorithm 1.

Algorithm 1 Deep Q-Learning

Input: θ
Output: a_i

```
for episode = 1, N do
    Observe s_0
    for i = 1, t do
        /*Action Selection*/
        Select a_i according pre-defined policy
        Observe s_{i+1}
        Calculate r_i by Equation (16)
        /*Q-network Update*/
        Output a_i
    end for
end for
```

4.2.2. Q-Network Update

Traditional Q-learning is a model-free reinforcement learning that solves the Bellman equation using asynchronous iterations. In Q-learning, $Q(s, a)$ represents the Q-value of a state–action pair (s, a). The value of $Q(s, a)$ can be considered as the expected payoff of the task, which takes action at state s, i.e., an edge server is chosen as the execution location for that task. At state s, the optimal policy can be formulated as:

$$\pi^*(s) = \arg\max_{a \in \mathbb{A}} Q^*(s, a) \tag{17}$$

where $Q^*(s, a)$ is the optimal Q-value of the state–action pair (s, a). Therefore, to obtain the optimal policy for current state s, we need to calculate the Q-value for all actions $a \in \mathbb{A}$. In the Q-learning algorithm, each $Q(s, a)$ can be updated with the following learning rule:

$$Q(s, a) = Q(s, a) + \alpha\big(R(s, a) + \gamma \max_{a' \in \mathbb{A}} Q(s', a') - Q(s, a)\big) \tag{18}$$

where $\alpha \in (0, 1]$ is the learning rate and γ is the discount parameter. Meanwhile, in the deep Q-learning algorithm, the Q-values are abstractly stored in the DNN. The system's state changes in each time slot, and the DNN must be adaptively updated.

Therefore, an essential part of the deep Q-learning algorithm is the DNN training based on the historical information stored in the experience replay memory $\mathbb{M}$ [47]. In the t-th time slot, we randomly selected a batch of training data samples from $\mathbb{M}$. The DNN with weights θ was updated by applying the Adam algorithm [48] to reduce the averaged cross-entropy loss, as

$$L(\theta_t) = -\frac{1}{|\mathbb{M}|}\sum_{\tau}^{\mathbb{M}}\left((\mathbf{a}_\tau^*)^\intercal \log f_{\theta_t}(\mathbf{s}_\tau) + (1-\mathbf{a}_\tau^*)^\intercal \log\left(1 - f_{\theta_t}(\mathbf{s}_\tau)\right)\right) \tag{19}$$

where $|\mathbb{M}|$ denotes the size of $\mathbb{M}$. For simplicity, the detailed update procedure of the Adam algorithm is omitted. The use of historical data can reduce the variance in θ_t, while the random sampling approach reduces the correlation among the training data, thus accelerating the convergence of $L(\theta)$. Meanwhile, as size of $|\mathbb{M}|$ is fixed and the memory space $\mathbb{M}$ is updated in each iteration, the DNN learns from the latest data samples generated by the recent strategies, which constantly improve the algorithm's decision. Overall, the detailed algorithm steps for the Q-network update are shown in Algorithm 2.

Algorithm 2 Q-Network Update

Input: $s_\tau, a_\tau, r_\tau, s_{\tau+1}, \mathbb{M}$
Output: θ
Store $(s_\tau, a_\tau, r_\tau, s_{\tau+1})$ in $\mathbb{M}[i \, mod \, |\mathbb{M}|]$
Sample $\mathbb{M}_\sim \subset \mathbb{M}$
for $(s_{\tau-1}^{(j)}, a_{\tau-1}^{(j)}, r_{\tau-1}^{(j)}, s_\tau^{(j)})$ in $\mathbb{M}_\sim$ **do**
    Calculate $Q(s_\tau^{(j)}, a_\tau^{(j)}; \theta')$
    Calculate target network's Q-value Q_{target}
    Compute the error between $Q(s_\tau^{(j)}, a_\tau^{(j)}; \theta')$ and Q_{target}
end for
$\theta = argmin_\theta L(\theta)$
Occasionally reset $\theta' = \theta$
Return θ

Based on the reinforcement settings and Q-network update, we present our deep reinforcement-learning-based dynamic computation-offloading strategy (DCOS) in the next section.

4.2.3. DCOS

The DQN-based dynamic computation offloading algorithm DCOS is proposed in Algorithm 3. Algorithm 3 includes two parts, reinforcement settings and Q-network updates. A detailed description of the reinforcement settings and Q-network update is given in Algorithms 1 and 2.

Algorithm 3 DCOS

Input: $\theta, \mathbb{M}$
Output: a_i
for episode = 1, N **do**
    Observe s_0
    $\mathbb{M} = \emptyset$
    Initialize $\mathbb{M}$
    for i = 1, t **do**
        /*Action Selection*/
        Select a_i according pre-defined policy
        Observe s_{i+1}
        Calculate r_i by Equation (16)
        /*Call Algorithm 2*/
        $\theta = QNetworkUpdate(s_i, a_i, r_i, s_{i+1}, \mathbb{M})$
        Output a_i
    end for
end for

In view of the computational complexity of Algorithm 3, the basic elements of the network model are defined in Section 3.1: the network contains n edge servers and m

types of services, and k task requests are issued in each time slot. The time complexity of Algorithm 3 mainly focuses on reinforcement learning settings and the Q-network update. As shown in Section 4.2.1, the main steps in reinforcement learning settings include obtaining the system status, action selection and reward calculation. For obtaining system, the time complexity is related to the size of the state, which is $O(2n + 3m)$. The time complexity of reward calculation is $O(n + m + k)$. Regarding the choice of actions, there are two cases, according to the value of ϕ. Considering these two cases simultaneously, the time complexity is $O(n)$. In the Q-network update, the time complexity is related to the size of the experience replay memory $\mathbb{M}$, which depends on the size of the state space $\mathbb{S}$ and action space $\mathbb{A}$. Therefore, the time complexity of this step is approximately $O(n^2)$. All steps are executed sequentially, so Algorithm 3 has polynomial time complexity.

5. Experiments

In this section, we first introduce relevant settings for the experiments in Section 5.1. The simulation experiments for the dynamic computation offloading algorithm proposed in this paper will be performed in Section 5.2.

5.1. Experimental Settings

In system model simulation settings, to illustrate the performance of the proposed strategy, we set an area of 500×500 m^2, which includes 50 randomly deployed edge servers. The time slot value was random in 20–30 ms, and the IoT device issue k task requests at each time slot. For all simulation parameters, we simplified some parameters, but kept key parameters directly involved in our algorithm's decision-making process. The key parameters are described in Table 1.

Table 1. Simulation parameters.

Parameters	Simulation Value	Description
$F.cap$	2 GHz	The processing capability of edge server
T_i	random in [20, 30] ms	The i-th time slot
n	50	The number of edge servers
k	30	The max number of task requests at each time slot
m	50	The number of service classes
$H.bytes$	random in [0.2, 5] MB	The capacity of service
$C.cir$	uniform in [2, 12] cycles/bit	The number of CPU circles required by the task
$C.bytes$	random in [0.5, 5] MB	The task capacity
B	500 MHz	The channel bandwidth
N_{noise}	-100 dBm	The background noise
ω_1	0.8	Weight of delay in total cost $\mathcal{C}$
ω_2	0.2	Weight of energy consumption in total cost $\mathcal{C}$
ϵ_{amp}	0.1 nJ/(bit $\times$ m^2)	The energy consumption constant of the transmit amplifier
e_{elec}	50nj/bit	The energy consumption constant of the transmission and receiver electronics

In the DQN algorithm, a deep neural network structure will significantly affect the efficiency of the algorithm. Our work shows that a simple four-layer network architecture, with one input layer, two hidden layers and one output layer, can achieve a better convergence performance. There were 100 and 60 hidden neurons, respectively, in these two hidden layers. The hidden layer used Relu as the activation function, and the sigmoid activation function was employed in the output layer to output the optimal action. The DQN parameters were configured as follows. The learning rate α equalled 0.01, the discount parameter γ was set to 0.9, and the training batch size and the size of experience replay memory were set to 100 and 200, respectively.

To illustrate and validate the effectiveness of our proposed DCOS algorithm, we focus on the following two metrics.

- The time delay for task-offloading, service migration, and task-processing.
- The energy consumption for task-offloading, service migration and task-processing.

To show the superiority of the proposed DCOS algorithm in terms of its delay and energy metrics, we compare it with two baseline schemes, as follows:

- Deep Deterministic Policy Gradient (DDPG) [33]: Task scheduling is only based on the task-offloading strategy, and the optimal offloading target node is output by the DQN network.
- Extensive Service Migration Model (ESM) [39]: The task is processed on the local node. If the node has not configured the related service, the system model performs the service migration according to the optimal policy related to the migration costs.

5.2. Performance Evaluation

In this section, to evaluate the effectiveness of the DCOS algorithm, we evaluate the algorithm's performance in terms of both task density and number of services. With the change in task density, Section 5.2.1 analyzes the performance of user request delays, device energy consumption and total consumption. As the number of services changes, Section 5.2.2 analyzes the performance of user request delays, device energy consumption and total consumption.

5.2.1. Influence of Task Density

We ran our experiments on several task densities—10, 25, 40, 55 tasks each second—to analyze the influence of different task densities on different algorithms. The task density varied from 10 to 55, representing a range from a low- to high-workload network state. In Figures 4 and 5, we plot a comparison of the energy consumption and delay for several task densities. A comparison of the total cost is shown in Figure 6.

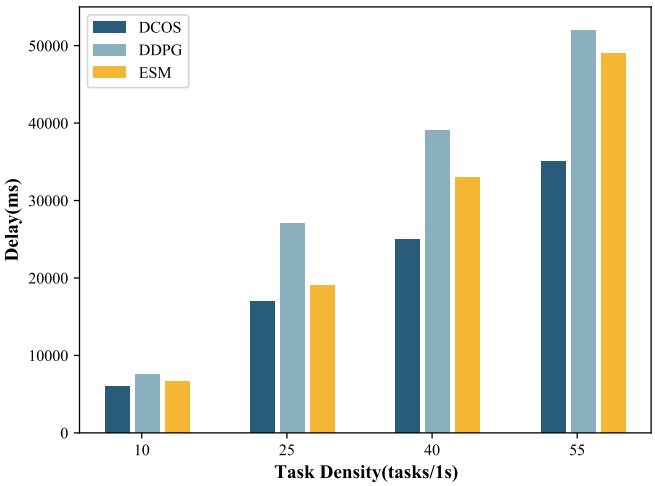


Figure 4. Delay comparison on different task densities.

As shown in Figure 4, when the task density is 15, due to the low task density, the current task is processed before the next task reaches the edge server. The computing resources of a single edge server can satisfy most of the task requests, and the necessity of collaboration is greatly reduced. Hence, the gap in the delay between the three collaboration strategies is not large. When the task density is 25, the computing resources of a single edge server cannot meet the requirements of the task requests. A collaboration strategy is needed to reduce the processing delay in tasks at this situation. We can observe that the performance of DCOS and ESM are closer because the ESM algorithm restricts the task execution location to local execution. In the long-term, the workload of each edge server is balanced, so there is little difference in the performance of the two algorithms under a low workload. The performance of the DDPG algorithm is worse because it cannot dynamically reconfigure the service. With the increase in task density and workload, the difference between the the time costs of both DDPG and ESM algorithms and those of DCOS gradually increases. When the network reaches saturation, the DCOS algorithm can

offload tasks to intermediate nodes with sufficient computing resources to reduce queuing delays. The parallel migration and offloading process enables the algorithm to complete the transmission of tasks and services at the most considerable time cost. When the task density reaches 55, the performance of DCOS clearly improves. Compared with DDPG, the delays in user requests decrease by 32.18%.The relative ESM delay in user requests is reduced by 27.16%.

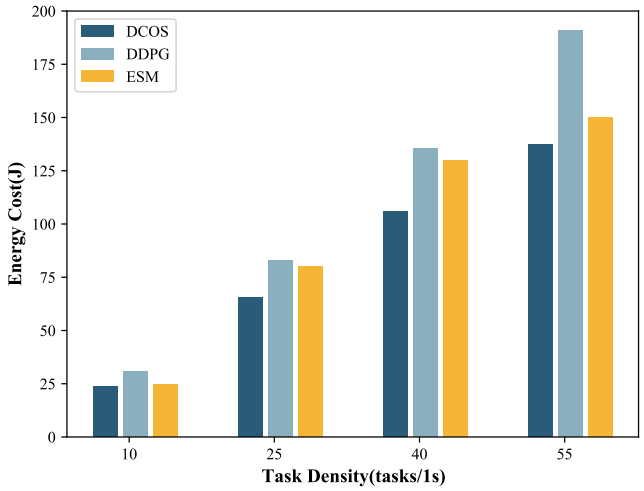


Figure 5. Energy consumption comparison for different task densities.

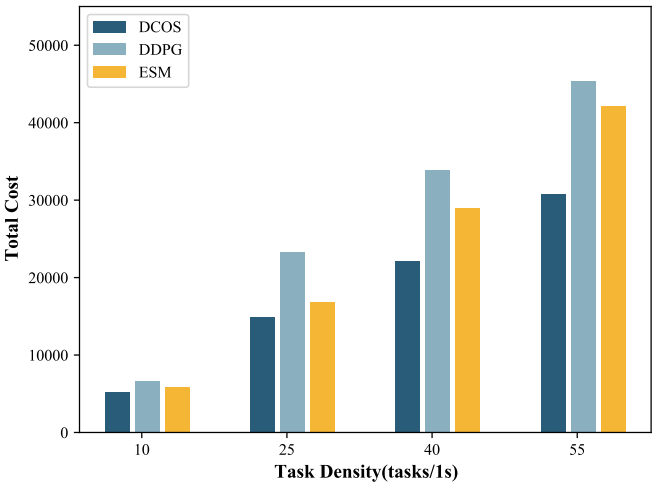


Figure 6. Total cost comparison for different task densities.

As shown in Figure 5, the energy consumption of the three algorithms increases with the increase in task density in a near-linear correlation, which is because all three algorithms inevitably need to transmit data, and the energy consumed by the transmission is independent of the operating status of the edge servers. It is worth mentioning that the task-processing delay is related to the operating status of the edge servers. When the workload of the edge network is over-saturated, the task-processing appears congested, so the delay is greater than the increase in energy consumption. In addition, since most of the tasks in this experiment are data-intensive, the amount of data transmitted during task offloading is larger than that during service migration, so the energy consumption of DDPG is larger than that of ESM. The DCOS algorithm combines the advantages of both algorithms and can flexibly adjust the decision according to the cost. Therefore it performs better than DDPG and ESM in terms of energy consumption. When the task density reaches 40, the energy consumption performance of DCOS is significantly improved compared with that of ESM, decreasing by 18.67%. When the task density reaches 40, the energy

consumption performance of DCOS is significantly improved compared with that of DDPG, decreasing by 28.21%.

The observation made in Figure 6 shows that the DDPG and ESM algorithms perform only within a very small range of task densities. However, our proposed DCOS algorithm remains stable when the task density increases. The results demonstrate that the DCOS algorithm can perform cost-effective computation-offloading in various situations. Due to the weight of the heavy delays in user requests, the trend in the data in Figures 4 and 6 is basically the same. The higher the task density, the better the DCOS performance.

5.2.2. Influence of Number of Services

On the edge networks, the types of services requested by tasks are diverse. However, due to the restricted resources of edge servers, only a limited number of services can be hosted. Therefore, we ran our experiments on several service numbers—10, 40, 70, 100—to analyze the influence of the different number of services on different algorithms. In this experiment, we set the task density to 55 to diversify the types of services requested by the task. In Figures 7 and 8, we plotted a comparison of the energy consumption and delay for several service numbers. A comparison of the total cost is shown in Figure 9.

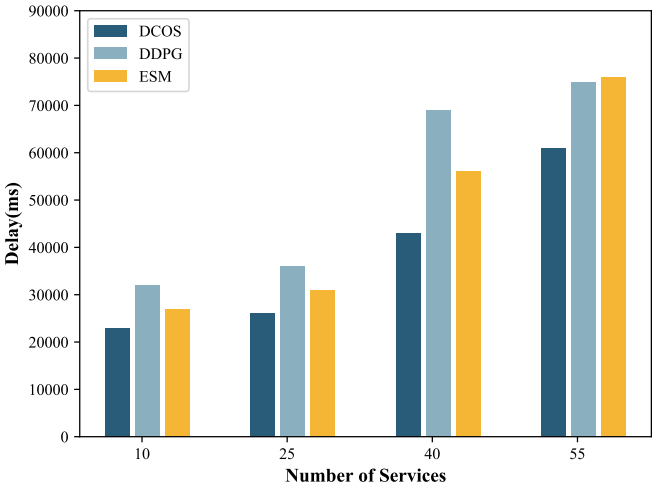


Figure 7. Comparison of delays in different service numbers.

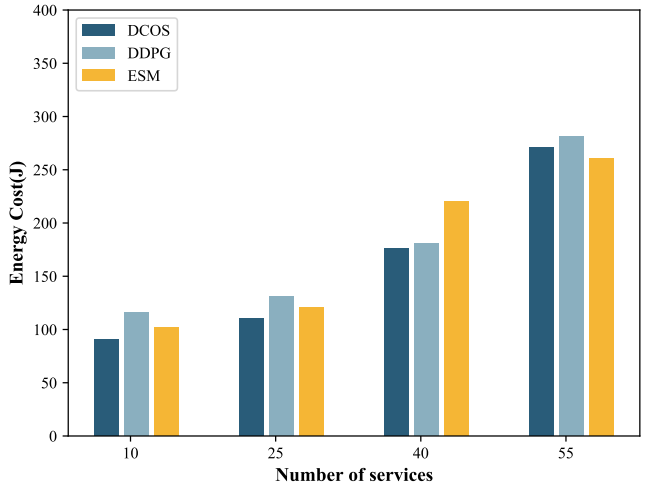


Figure 8. Comparison of energy consumption of different service numbers.

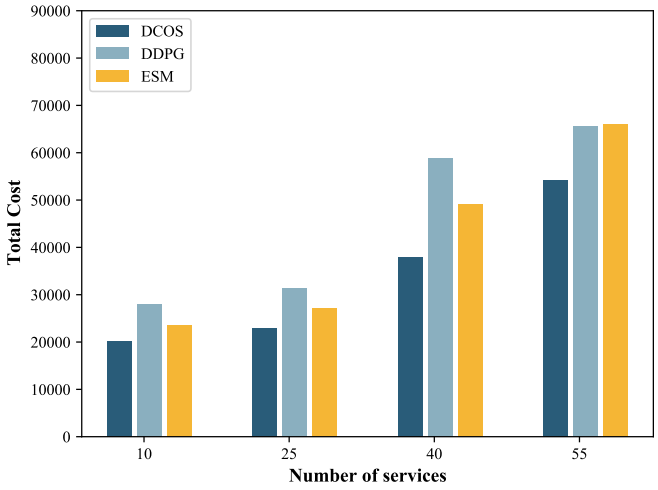


Figure 9. Total cost comparison for different service numbers.

As shown in Figure 7, the increase in the number of services results in a reduction in the probability of configuring the corresponding services in adjacent servers. This means that services need to be transmitted from a more distant server, and the increase in the transmission distance leads to an increase in the delay in user requests. When the service types are 10 and 40, the edge network can configure that number of services, so there is little difference in the delay between the three algorithms. When the number of services is 70, the probability that the neighboring servers are configured with the related services further decreases; therefore, the increase in delay becomes larger. When the number of services is 100, not all services can be fully configured, due to the resource constraints of the edge servers. When the service requested by the task is not configured in the edge network, the task cannot be processed, i.e., an off-target situation occurs, leading to a spike in delays. However, benefiting from our strategy, we are able to significantly reduce the delay when collaborating among edge servers, so the delay in the performance of user requests is still better than that of the other two algorithms.

As shown in Figure 8, when the number of services is 10 and 40, the performance of the three algorithms in terms of the energy consumption of IoT devices is similar to that of the delay performance. However, with the increase in the number of services, because the decision of the DCOS algorithm pays more attention to the reduction in delay, the increase in the energy consumption of IoT devices is more significant than the increase in user request delays. Our strategy can efficiently reduce the delay, but the energy consumption of offloading tasks and the migration of services to intermediate nodes is inevitable. In terms of energy consumption, our strategy is not significantly better than that of DDPG and ESM.

Limited by the resources of the edge servers, the edge network can only host a certain number of services; if the number of services exceeds the threshold, this will lead to off-target situations, which cause delays in user requests and the energy consumption of IoT devices to become larger. Through Figure 9, we can observe that the DCOS algorithm still outperforms DDPG and ESM in terms of the total cost, because our algorithm is more concerned with reducing textcolorbluethe delays in user requests, and can make flexible decisions that allow for better collaboration between edge servers, reducing the total cost. This demonstrates the effectiveness and flexibility of the DCOS algorithm.

6. Conclusions and Future Works

This paper mainly studies the problem of computation offloading in edge networks. To fully exploit the potential of collaboration edge servers, a dynamic computation offloading strategy based on deep reinforcement learning is proposed. By selecting intermediate nodes with sufficient computing resources as task execution locations, the processing delays in the task are significantly reduced and data transmission is completed at the lowest possible

cost through parallel migration and offloading processes. Then, we designed an algorithm based on deep reinforcement learning to efficiently select the optimal intermediate nodes. Simulation experiments show that our approach can fully utilize the computing resources of edge servers, allowing them to make better decisions, and leading to a lower energy consumption by IoT devices and delays in user requests under the same experimental settings. In addition, the stability and flexibility of our algorithm are demonstrated by comparison with two baseline algorithms. However, there is still a challenge that needs to be overcome in future work. Since our algorithm pays more attention to reducing the the delay in user requests, it does not perform well in terms of the energy consumption of IoT devices. In particular, when deviations from the target occur in the network, our algorithm does not significantly reduce the delay in user requests or the energy consumption of IoT devices. To overcome this challenge, our next work will focus on how to initialize the service configuration of the network to further improve the effectiveness of the algorithm.

Author Contributions: All authors listed in this paper contributed equally, as follows. Y.B. contributed significantly to analysis and manuscript preparation. X.L. was responsible for the methodology design. X.W. was responsible for visualization/data presentation. Z.Z. was responsible for ensuring that the descriptions are accurate and agreed by all authors. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program of China (2019YFB2101803), and by the National Natural Science Foundation of China (61772479 and 42050103).

Acknowledgments: I would like to give my heartfelt thanks to all the people who have ever helped me in this paper.

Conflicts of Interest: We declare that we have no competing fnancial interest or personal relationship that could have appeared to infuence the work reported in this paper.

Notations

Symbol	Description
F	Edge server set
n	Number of edge servers
$F_i.id$	The unique identifier of edge server i
$F_i.loc$	The location of edge server i
$F_i.cap$	The processing capability of edge server i
$F_i.pt$	The task list processed on edge server
H	Service set
m	Number of service classes
$H_i.id$	The unique identifier of service i
$H_i.bytes$	The capacity of service i
T	Time period
T_i	The i-th time slot in T
C	Task request set
k	Number of task requests
$C_i.id$	The unique identifier of task i
$C_i.loc$	The location of task i
$C_i.bytes$	The capacity of task i
$C_i.H_j$	The service type required by the task i
$C_i.cir$	The number of CPU circles required by the task i.
B	Channel bandwidth
N_{noise}	The background noise
e_{elec}	The energy consumption constant of the transmission and receiver electronics
ϵ_{amp}	The energy consumption constant of the transmit amplifier
d_{total}	Total delay
e_{total}	Total energy consumption

$\mathcal{C}$	Total cost
ω_1	Weight of delay in $\mathcal{C}$
ω_2	Weight of energy consumption in $\mathcal{C}$
s_i	System state during T_i
a_i	The best action to output after observing state s_i
r_i	Reward during T_i
$\mathbb{S}$	State space
$\mathbb{A}$	Action space
$Q(s,a)$	The Q-value of state-action pair (s,a)
$\mathbb{M}$	Experience replay memory space
α	Learning rate
γ	Discount parameter
$L(\theta)$	Loss function of the DNN
θ	Weights of the DNN

References

1. Wang, K.; Yang, Z.; Liang, B.; Ji, W. An intelligence optimization method based on crowd intelligence for IoT devices. *Int. J. Crowd Sci.* **2021**, *5*, 218–227. [CrossRef]
2. Tang, J.; Wu, S.; Wei, L.; Liu, W.; Qin, T.; Zhou, Z.; Gu, J. Energy-Efficient Sensory Data Collection Based on Spatiotemporal Correlation in IoT Networks. *Int. J. Crowd Sci.* **2022**, *6*, 34–43. [CrossRef]
3. Ma, X.; Li, Q.; Zou, L.; Peng, J.; Zhou, J.; Chai, J.; Jiang, Y.; Muntean, G.M. QAVA: QoE-Aware Adaptive Video Bitrate Aggregation for HTTP Live Streaming Based on Smart Edge Computing. *Trans. Broadcast.* **2022**, *68*, 661–676. [CrossRef]
4. Lee, S.H. Real-time edge computing on multi-processes and multi-threading architectures for deep learning applications. *Microprocess. Microsyst.* **2022**, *92*, 104554. [CrossRef]
5. Bonomi, F. Connected vehicles, the internet of things, and fog computing. In Proceedings of the The Eighth ACM International Workshop on Vehicular Inter-Networking (VANET), Las Vegas, NV, USA, 19–23 September 2011; pp. 13–15.
6. Shakarami, A.; Shahidinejad, A.; Ghobaei-Arani, M. An autonomous computation offloading strategy in Mobile Edge Computing: A deep learning-based hybrid approach. *J. Netw. Comput. Appl.* **2021**, *178*, 102974. [CrossRef]
7. Osei-Mensah, E.; Thabet, S.K.S.; Luo, C.; Asiedu-Ayeh, E.; Bamisile, O.; Nyantakyi, I.O.; Adun, H. A Novel Distributed Media Caching Technique for Seamless Video Streaming in Multi-Access Edge Computing Networks. *Appl. Sci.* **2022**, *12*, 4205. [CrossRef]
8. Chen, S.; Zheng, Y.; Lu, W.; Varadarajan, V.; Wang, K. Energy-optimal dynamic computation offloading for industrial iot in fog computing. *Trans. Green Commun. Netw.* **2019**, *4*, 566–576. [CrossRef]
9. Adhikari, M.; Mukherjee, M.; Srirama, S.N. DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing. *Internet Things J.* **2019**, *7*, 5773–5782. [CrossRef]
10. Liu, L.; Chang, Z.; Guo, X. Socially aware dynamic computation offloading scheme for fog computing system with energy harvesting devices. *Internet Things J.* **2018**, *5*, 1869–1879. [CrossRef]
11. Zhang, G.; Shen, F.; Yang, Y.; Qian, H.; Yao, W. Fair task offloading among fog nodes in fog computing networks. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
12. Qin, M.; Cheng, N.; Jing, Z.; Yang, T.; Xu, W.; Yang, Q.; Rao, R.R. Service-oriented energy-latency tradeoff for iot task partial offloading in mec-enhanced multi-rat networks. *Internet Things J.* **2020**, *8*, 1896–1907. [CrossRef]
13. Bozorgchenani, A.; Tarchi, D.; Corazza, G.E. Centralized and distributed architectures for energy and delay efficient fog network-based edge computing services. *Trans. Green Commun. Netw.* **2018**, *3*, 250–263. [CrossRef]
14. Yuan, X.; Xie, Z.; Tan, X. Computation Offloading in UAV-Enabled Edge Computing: A Stackelberg Game Approach. *Sensors* **2022**, *22*, 3854. [CrossRef] [PubMed]
15. Shamsadini, A.; Entezari-Maleki, R. Time-aware MDP-based Service Migration in 5G Mobile Edge Computing. In Proceedings of the 2022 27th International Computer Conference, Computer Society of Iran (CSICC), Tehran, Iran, 23–24 February 2022; pp. 1–5.
16. Chen, X.; Bi, Y.; Chen, X.; Zhao, H.; Cheng, N.; Li, F.; Cheng, W. Dynamic Service Migration and Request Routing for Microservice in Multi-cell Mobile Edge Computing. *Internet Things J.* **2022**, *9*, 13126–13143. [CrossRef]
17. Xu, M.; Zhou, Q.; Wu, H.; Lin, W.; Ye, K.; Xu, C. PDMA: Probabilistic service migration approach for delay-aware and mobility-aware mobile edge computing. *Softw. Pract. Exp.* **2022**, *52*, 394–414. [CrossRef]
18. Xu, J.; Ma, X.; Zhou, A.; Duan, Q.; Wang, S. Path selection for seamless service migration in vehicular edge computing. *Internet Things J.* **2020**, *7*, 9040–9049. [CrossRef]
19. Labriji, I.; Meneghello, F.; Cecchinato, D.; Sesia, S.; Perraud, E.; Strinati, E.C.; Rossi, M. Mobility aware and dynamic migration of mec services for the internet of vehicles. *Trans. Netw. Serv. Manag.* **2021**, *18*, 570–584. [CrossRef]

20. Li, C.; Zhu, L.; Li, W.; Luo, Y. Joint edge caching and dynamic service migration in SDN based mobile edge computing. *J. Netw. Comput. Appl.* **2021**, *177*, 102966. [CrossRef]
21. Yuan, Q.; Li, J.; Zhou, H.; Lin, T.; Luo, G.; Shen, X. A joint service migration and mobility optimization approach for vehicular edge computing. *Trans. Veh. Technol.* **2020**, *69*, 9041–9052. [CrossRef]
22. Li, J.; Chen, L.; Chen, J. Enabling technologies for low-latency service migration in 5G transport networks. *J. Opt. Commun. Netw.* **2021**, *13*, A200–A210. [CrossRef]
23. Liu, Z.; Xu, X. Latency-aware service migration with decision theory for Internet of Vehicles in mobile edge computing. *Wirel. Netw.* **2022**. [CrossRef]
24. Chen, S.; Tang, B.; Wang, K. Twin delayed deep deterministic policy gradient-based intelligent computation offloading for IoT. *Digit. Commun. Netw.* 2022, *in press*. [CrossRef]
25. Wang, S.; Urgaonkar, R.; Zafer, M.; He, T.; Chan, K.; Leung, K.K. Dynamic service migration in mobile edge computing based on markov decision process. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1272–1288. [CrossRef]
26. Liu, J.; Ji, W. Evolution of Agents in the Case of a Balanced Diet. *Int. J. Crowd Sci.* **2022**, *6*, 1–6. [CrossRef]
27. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
28. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Hassabis, D. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
29. Wang, H.; Xu, H.; Huang, H.; Chen, M.; Chen, S. Robust task offloading in dynamic edge computing. *Trans. Mob. Comput.* **2021**, *22*, 500–514. [CrossRef]
30. Zhou, J.; Tian, D.; Sheng, Z.; Duan, X.; Shen, X. Distributed task offloading optimization with queueing dynamics in multi-agent mobile-edge computing networks. *Internet Things J.* **2021**, *8*, 12311–12328. [CrossRef]
31. Wang, J.; Hu, J.; Min, G.; Zomaya, A.Y.; Georgalas, N. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *Trans. Parallel Distrib. Syst.* **2020**, *32*, 242–253. [CrossRef]
32. Tran-Dang, H.; Kim, D.S. Frato: Fog resource based adaptive task offloading for delay-minimizing iot service provisioning. *Trans. Parallel Distrib. Syst.* **2021**, *32*, 2491–2508. [CrossRef]
33. Qinghua, Z.; Ying, C.; Jingya, Z.; Yong, L. Computation offloading Optimization in Edge Computing based on Deep Reinforcement Learning. In Proceedings of the 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE), Harbin, China, 25–27 December 2020; pp. 1552–1558.
34. Kim, T.; Sathyanarayana, S.D.; Chen, S.; Im, Y.; Zhang, X.; Ha, S.; Joe-Wong, C. Modems: Optimizing edge computing migrations for user mobility. *J. Sel. Areas Commun.* **2022**. [CrossRef]
35. Liang, Z.; Liu, Y.; Lok, T.M.; Huang, K. Multi-cell mobile edge computing: Joint service migration and resource allocation. *Trans. Wirel. Commun.* **2021**, *20*, 5898–5912. [CrossRef]
36. Li, C.; Zhang, Y.; Gao, X.; Luo, Y. Energy-latency tradeoffs for edge caching and dynamic service migration based on DQN in mobile edge computing. *J. Parallel Distrib. Comput.* **2022**, *166*, 15–31. [CrossRef]
37. Zhang, C.; Liu, Z.; Gu, B.; Yamori, K.; Tanaka, Y. A deep reinforcement learning based approach for cost-and energy-aware multi-flow mobile data offloading. *IEICE Trans. Commun.* **2018**, *101*, 1625–1634. [CrossRef]
38. Tang, Z.; Zhou, X.; Zhang, F.; Jia, W.; Zhao, W. Migration modeling and learning algorithms for containers in fog computing. *Trans. Serv. Comput.* **2018**, *12*, 712–725. [CrossRef]
39. Park, S.W.; Boukerche, A.; Guan, S. A novel deep reinforcement learning based service migration model for mobile edge computing. In Proceedings of the 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), Prague, Czech Republic, 14–16 September 2020; pp. 1–8.
40. Jiao, Y.; Wang, C. A Blockchain-Based Trusted Upload Scheme for the Internet of Things Nodes. *Int. J. Crowd Sci.* **2022**, *6*, 92–97. [CrossRef]
41. Dinh, H.T.; Lee, C.; Niyato, D.; Wang, P. A survey of mobile cloud computing: Architecture, applications, and approaches. Wireless communications and mobile computing. *Wirel. Commun. Mob. Comput.* **2013**, *13*, 1587–1611. [CrossRef]
42. Willis, D.; Dasgupta, A.; Banerjee, S. Paradrop: A multi-tenant platform to dynamically install third party services on wireless gateways. In Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture, Maui, HI, USA, 11 September 2014; pp. 43–48.
43. Bittencourt, L.F.; Lopes, M.M.; Petri, I.; Rana, O.F. Towards virtual machine migration in fog computing. In Proceedings of the 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Krakow, Poland, 4–6 November 2015; pp. 1–8.
44. Wang, S.; Urgaonkar, R.; Zafer, M.; He, T.; Chan, K.; Leung, K.K. Dynamic service migration in mobile edge-clouds. In Proceedings of the 2015 IFIP Networking Conference (IFIP Networking), Toulouse, France, 20–22 May 2015; pp. 1–9.
45. Heinzelman, W.R.; Chandrakasan, A.; Balakrishnan, H. Energy-efficient communication protocol for wireless microsensor networks. Wireless communications and mobile computing. In Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, Maui, HI, USA, 4–7 January 2000; p. 10.
46. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *Commun. Surv. Tutorials* **2017**, *19*, 2322–2358. [CrossRef]

47. Rouzbahani, H.M.; Karimipour, H.; Lei, L. Optimizing scheduling policy in smart grids using probabilistic Delayed Double Deep Q-Learning (P3DQL) algorithm. *Sustain. Energy Technol. Assessments* **2022**, *53*, 102712. [CrossRef]
48. Reddi, S.J.; Kale, S.; Kumar, S. On the convergence of adam and beyond. *arXiv* **2019**, arXiv:1904.09237.

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.