applied sciences

MDPI

Article

DRL-Based Dependent Task Offloading Strategies with Multi-Server Collaboration in Multi-Access Edge Computing

Biying Peng [1], Taoshen Li [2,1,*] and Yan Chen [1,*]

[1] Department of Computer and Electronic Information, Guangxi University, Nanning 530004, China
[2] Department of information and Engineering, Nanning University, Nanning 530001, China
* Correspondence: tshli@gxu.edu.cn (T.L.); cy@gxu.edu.cn (Y.C.)

Abstract: Many applications in Multi-access Edge Computing (MEC) consist of interdependent tasks where the output of some tasks is the input of others. Most of the existing research on computational offloading does not consider the dependency of the task and uses convex relaxation or heuristic algorithms to solve the offloading problem, which lacks adaptability and is not suitable for computational offloading in the dynamic environment of fast fading channels. Therefore, in this paper, the optimization problem is modeled as a Markov Decision Process (MDP) in multi-user and multi-server MEC environments, and the dependent tasks are represented by Directed Acyclic Graph (DAG). Combined with the Soft Actor–Critic (SAC) algorithm in Deep Reinforcement Learning (DRL) theory, an intelligent task offloading scheme is proposed. Under the condition of resource constraint, each task can be offloaded to the corresponding MEC server through centralized control, which greatly reduces the service delay and terminal energy consumption. The experimental results show that the algorithm converges quickly and stably, and its optimization effect is better than existing methods, which verifies the effectiveness of the algorithm.

Keywords: multi-access edge computing; directed acyclic graph; deep reinforcement learning; offloading strategy

check for updates

Citation: Peng, B.; Li, T.; Chen, Y. DRL-Based Dependent Task Offloading Strategies with Multi-Server Collaboration in Multi-Access Edge Computing. *Appl. Sci.* **2023**, *13*, 191. https://doi.org/10.3390/app13010191

Academic Editor: Giacomo Fiumara

Received: 7 December 2022
Revised: 19 December 2022
Accepted: 20 December 2022
Published: 23 December 2022

Copyright: © 2022 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

Recently, with the proliferation of wireless and Internet of Things technologies, numerous innovative applications have emerged, such as autopilot, Wise Information Technology of Med (WITMED), Augmented Reality (AR), Mixed Reality (MR), and Virtual Reality (VR), that demand intensive resources and massive computing power. Therefore, how to handle the rapid growth of bandwidth-intensive user requests while providing a higher-quality experience has become one of the biggest challenges [1]. Thus, Multi-access Edge Computing (MEC) was proposed to satisfy the heavy demand for computing resources for applications. The technology of task offloading in MEC can offload computation-intensive tasks from user equipment (UE) to MEC hosts.

The offloading problem is NP-hard [2], so most studies use heuristic or convex optimization algorithms. However, both methods may fall into local optima and cannot guarantee good performance. With the increasing complexity of MEC architecture and applications, new scenarios emerge. These two algorithms need to adapt to new scenarios constantly. Moreover, once the wireless channel conditions change or the available computing capacity of the edge server changes due to the demand of the background application, the optimization problem needs to be solved again. DRL can achieve flexible and adaptive offloading in the MEC environment to maximize numerical rewards. Some works have adopted Deep Reinforcement Learning (DRL) to deal with the task offloading problem in MEC, but they only pay attention to the coarse-grained offloading problem, without considering the task dependencies. Ignoring the dependencies between tasks fails to meet the actual requirements and severely affects Quality-of-Service (QoS).

In addition, many works have only considered the offloading problem in the single-server scenario but have not studied the multiserver MEC scenario. In practice, a densely deployed multiple MEC server can perform uninterrupted task offloading to meet the service demand [3].

To cover these research gaps, we proposed a Soft Actor–Critic-based Dependent Task Offloading (SACDTO) strategy; designed a new objective function in the multi-user, multi-server MEC environment; and modeled the optimization problem as a Markov Decision Process (MDP). The dependent tasks are represented by Directed Acyclic Graphs (DAG), where vertices and edges denote tasks and their dependency, respectively. In addition, we utilize the Soft Actor–Critic (SAC) method based on a maximum entropy reinforcement learning framework. Under resource constraints, we can offload tasks to the specified server number. A large number of simulation experiments show that SACDTO can drastically reduce the service delay and terminal energy consumption. The SACDTO strategy learns to make proper offloading decisions by directly interacting with the environment. The chief contributions of this paper are summarized as follows:

- We design an MDP to accurately model the dependent task offloading problem with well-designed state space, action space, and reward function. Aiming at the above problems, we propose an intelligent computing offloading algorithm based on discrete SAC, which can adapt to the dynamic network environment and has the advantages of high stability and high sample utilization. The SACDTO strategy with the maximum entropy objective has a higher exploration ability to learn a continuous action space.
- We extend the application scenario to multiple edge servers and study the computational offloading problem of various mobile users offloading tasks to multiple MEC servers through the base station. As multiple servers are deployed around mobile users, task offloading is no longer a simple binary offloading decision, and there are not only two processing methods: local execution or edge offloading. Instead, specific consideration should be given to whether task offloading should be performed and the number of the MEC server responsible for a task's execution after offloading.
- We use the collaboration between the cloud, edge, and terminal to realize the training of a neural network with the massive computing power of cloud, and the trained scheduler can offload tasks to MEC servers for calculation.
- We transform the original DAG into a series of embeddings that contain task profiles and dependency information for the application, which are transformed into inputs to the neural network. In addition, we conduct plenty of simulation experiments using synthetic DAG, analyze the convergence curves of each strategy, and discuss the time delay and energy consumption when the number of tasks and the communication to calculation ratio are different, which correspond to the characteristics of real applications. The results show that the proposed method outperforms other comparison algorithms in dynamic MEC scenarios.

2. Related Work

There is a lot of research on task offloading in MEC, which can be divided into three categories. First of all, for the dependent task offloading problem, some studies use the task call graph to model the complex dependencies between components in mobile applications [4,5], and some also use DAG [6–8] to represent the relationships between tasks. Specifically, Fan et al. [4] converts the task decision problem of cost minimization into the shortest path problem, and uses the classical Lagrangian relaxation-based aggregate cost algorithm to approximate the problem. Zhang et al. [5] combined a set of fine-grained tasks to form a common topology, which expanded the tasks into a general task map. Mao et al. [6] proposed a graph mapping task offloading model based on Deep Reinforcement Learning (DRL), which converts DAG tasks into topological sequences according to custom priorities and then maps them into offloading decisions. Chen et al. [7] proposed ACED, a multi-dependent task computing offloading algorithm based on DAG, which is an actor–critic mechanism with two embedded layers. Leng et al. [8] proposed a reinforcement learning

method based on a graph convolution network, which regards the task set as a directed acyclic graph and uses a graph convolution network to extract features from tasks.

DRL is more suitable for online offloading in a fast fading channel environment. At present, convex relaxation methods [9] or heuristic local search methods [4,10] are mostly used for task offloading. However, both methods have their limitations and may fall into local optima. Moreover, it is impractical to re-solve the optimization problem once the external environment changes. DRL-based approaches include value-based and policy-based approaches. Frequently used value-based DRL methods include Deep Q Learning Network (DQN) [9], Double DQN [11], Dueling DQN [12], and Double Dueling DQN (D3QN) [13]. However, when the number of wireless devices grows exponentially, DQN-based approaches are expensive. To solve this problem, some works have applied policy-based methods, such as the actor–critic method [14,15]. Liu et al. [14] used the actor–critic model to solve the offloading problem of fine-grained tasks. In addition, the Proximal Policy Optimization (PPO) algorithm [16,17] has a good effect, and it can realize both discrete control and continuous control at the same time. However, PPO has the problem of sample inefficiency, which requires a huge amount of sampling to learn, which is unacceptable for practical application scenarios. Among them, Li et al. [16] studied the offloading problem in the multi-server and multi-user scenario, and Wang et al. [17] used the PPO algorithm to make decisions on the dependent tasks' offloading. Some studies also use DDPG to solve the offloading problem [18,19]. Compared with PPO, this algorithm has higher sampling efficiency. As a deterministic strategy, DNN is used to directly construct the optimal mapping strategy from the input state to the output action. SAC uses a stochastic policy, which has advantages compared to the deterministic policy, as well as good stability and high sampling efficiency. Some recent research has been using SAC to solve the computational offloading problem. Sun et al. [20] proposed centralized SAC offloading and decentralized SAC offloading algorithms. Liu et al. [21] proposed a DAG task offloading method in MEC with inter-user cooperation and used a Quantified Soft Actor–Critic (QSAC) algorithm to solve the target problem. We summarize the pros and cons of these approaches in Table 1.

Table 1. Comparison of Approaches.

Approaches	Advantages	Disadvantages
Convex relaxation approaches [9] or heuristic local search approaches [4,10]	Closer to the optimal offloading solution.	It is easy to fall into the local optima, and it is necessary to re-solve the optimization problem when the external environment changes.
DQN-based approaches [9–13]	Suitable for dynamic environments.	When the number of wireless devices grows exponentially, these approaches are expensive.
PPO-based approaches [16,17]	Good effect, and it can realize discrete control and continuous control.	The sample efficiency is low and requires a large number of samples, which is not suitable for actual application scenarios.
DDPG-based approaches [18,19]	High efficiency.	The explorability, stability, and robustness of this method are not good enough.

Since the computing capacity of the cloud server is far greater than that of the MEC server, and it has enough resources to meet the peak demand of user requests [22], it can be considered that the cloud and MEC can cooperate to complete the offloading task. Some tasks with high computational complexity still need to be processed by the cloud server. For example, we put the training of the offloading scheduler on the cloud for execution. MEC systems mentioned in the literature allow Cloud and edge collaboration [19,20,23–27], and mobile devices can adaptively offload dependent tasks to MEC or Cloud. He et al. [24] studied a multi-layer task offloading framework for MEC, which not only uses the collaboration between Cloud and MEC but also offloads tasks to other mobile devices through D2D links. Chen et al. [25] considered an MEC dependency sensing offloading scheme based on edge cloud collaboration under the constraint of task dependence.

3. System Model and Problem Formulation

The specific objective of this study was to propose an intelligent task offloading scheme in a multi-user and multi-server MEC environment that can offload each task to the corresponding MEC server through centralized control to reduce the service delay and terminal energy consumption.

SACDTO is a DRL-based offloading framework integrated into the MEC platform defined by ETSI [28]. As shown in Figure 1, the overall design of the SACDTO architecture is depicted. This system consists of three levels: the UE level, MEC level, and cloud level.

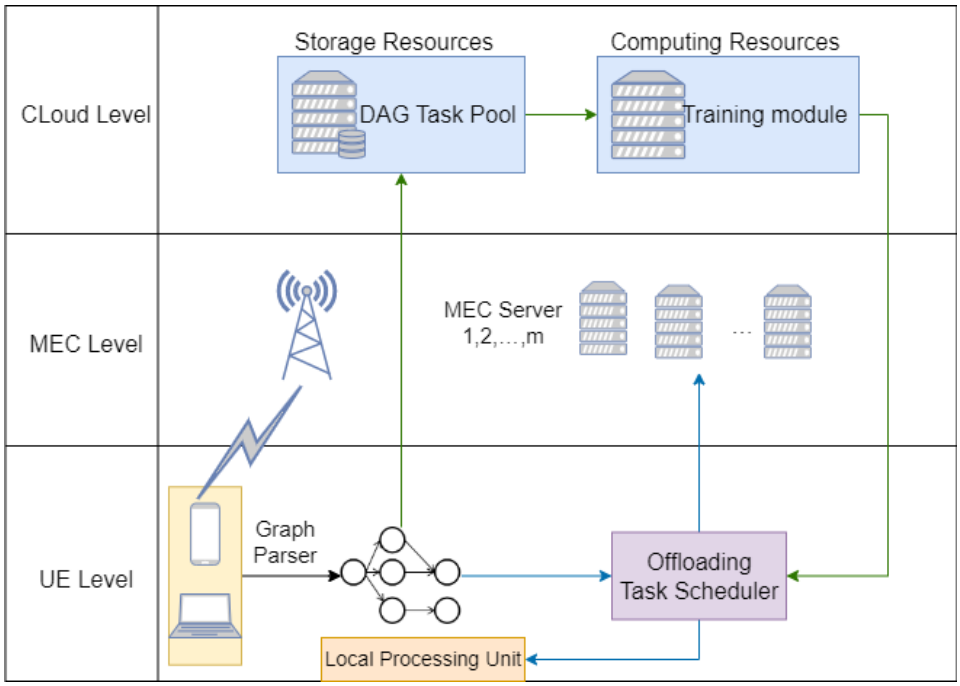


Figure 1. DRL-based MEC Offloading Scheme.

The system model is a multi-user and multi-server application scenario. There are N tasks and M MEC servers, and the task data to be offloaded is transmitted between the MEC server and the terminal device over the wireless communication link. This paper assumes that each terminal device can execute or offload the task locally, the task can only be offloaded to one MEC server for calculation, and each terminal device is within the range of wireless connection. Because the computing capacity of each MEC server is limited, it cannot accept the offloading requests of each terminal simultaneously. For the convenience of understanding, the mathematical notations are summarized in Table 2.

Table 2. Mathematical notations.

Notations	Definition
$\pi(\cdot \mid \cdot)$	Offloading policy
ω_i	The number of clock cycles required to process each bit of data
D	Replay buffer
P_{send}, P_{rec}	The transmitted and received power
ω	The parameter that determines the send or receive rate
$O_{1:i}$	Set of offloading decisions from task v_1 to v_i
ξ	Directed acyclic graph
β_1, β_2	The weight coefficient of energy consumption and delay ratio
$\theta 1$, $\theta 2$, ϕ	Parameters of SACDTO
α	The temperature parameter

Table 2. *Cont.*

Notations	Definition
a_i	The offloading action of task v_i
M	MEC server collection
T_i^{loc}, T_i^s	The local and MEC server computation delay of the task v_i
${f_m}^s$	The CPU clock speed of the MEC server numbered m
f_i^{loc}	The CPU clock speed of the mobile device where task v_i is located
$R_i(\omega)$	Sending rate or receiving rate
${C_m}^{MEC}$	Total computing resources of the MEC server numbered m
d_i	The data size of task v_i
C_i	The total number of clock cycles required by task v_i
$Finish_T_i$	The finish time of task v_i
$Finish_T_i^{loc}$	Local finish time of task v_i
$Finish_T_i^{trans}(\omega)$	The finish time of upload and download task v_i
$Finish_T_{pre}$	The finish time of the previous task
$Finish_T_{i,m}{}^{ser}$	The finish time of task v_i on the MEC server numbered m
$Avail_T_i^{loc}$	The CPU idle time of the earliest available task on the local processor
$Avail_T_i^{trans}(\omega)$	The earliest available time of the transmission link
$Avail_T_{i,m}{}^{ser}$	The earliest available CPU idle time of the MEC server

3.1. MEC Architecture

The user device layer includes a variety of mobile devices, a Graph Parser (GP), and a scheduler for the offloading tasks. In the MEC layer, multiple servers act as its computing resources. In the cloud layer, the task graph pool of DAG and the module responsible for training the offloading decision scheduler are included.

The architecture can implement the following process: (1) collect multiple tasks with dependencies from the mobile device and input these tasks to the graph parser; (2) computation-intensive tasks are converted into DAG, and then these DAGs are put into the scheduler training module of the cloud for DRL training; (3) the trained network parameters are sent back to the mobile device; (4) the trained neural network runs forward propagation to make offloading decisions for the task. The task can be processed by the MEC or executed by the local processing unit.

3.2. System Model

Different devices run tasks that cause different delays. This paper considers two cases: the local execution of the current task and it is offloaded to the MEC server. Since the strategy is to make offloading decisions for tasks with dependencies, the finish time of the previous tasks has an impact on the subsequent tasks. Therefore, this section will discuss the calculation delay, transmission delay, and task finish time calculation.

Let T_i^{loc} denote the local computation delay and $T_i{}^s$ denote the MEC server computation delay of the task v_i. In addition, ${f_m}^s$ and f_i^{loc} are the CPU clock speed of the MEC server numbered m and the mobile device where task v_i is located, respectively. ϖ_i is the number of clock cycles required to process each bit of data of task v_i, and C_i is the total number of clock cycles required by task v_i.

$$T_i^{loc} = \frac{C_i}{f_i^{loc}} = \frac{d_i \times \varpi_i}{f_i^{loc}},\ T_i^s = \frac{C_i}{{f_m}^s} = \frac{d_i \times \varpi_i}{{f_m}^s}. \tag{1}$$

Before calculating the delay of data offloading on mobile devices, the upload rate should be calculated according to Shannon's formula, which can be defined as:

$$R_i(\omega) = B \times \log_2\{1 + \frac{[\omega P_{send} + (1-\omega)P_{rec}] \times H_i^t}{\sigma^2}\} \tag{2}$$

where B is the wireless channel bandwidth between the mobile device and the edge cloud, and P_{send} and P_{rec} are the transmitted and received power, respectively. $\omega \in \{0, 1\}$, where, when $\omega = 1$, $R_i(\omega)$ represents the sending rate, and when $\omega = 0$, $R_i(\omega)$ represents the receiving rate.

H_i^t is the wireless channel gain between the device and the base station in time slot t, and σ^2 is the noise power consumption. Therefore, the value of ω in $T_i^{trans}(\omega)$ is 1 and 0, which, respectively, represent the upload delay or download delay between the mobile device and the MEC server. The transmission delay can be defined as:

$$T_i^{trans}(\omega) = \frac{d_i}{R_i(\omega)} \tag{3}$$

where M is the set of MEC server numbers, and the size of this set depends on the number of servers. Let $O_{1:n} = [a_1, a_2, \ldots, a_k, \ldots, a_n]$ denote the offloading decision for all tasks, $a_k \in \{0, M\}$. Specially, $a_k = 0$ means that task v_k is scheduled to the local processor. a_k is equal to any number other than 0, indicating that task v_k was offloaded to the MEC host numbered with that number. Assuming that the previous task of task v_i is v_k, it means that there exists a directed edge between v_k and v_i, which can be defined as $(v_k, v_i) \in \varepsilon$. Thus, the finish time of task v_k will have an impact on the finish time of task v_i. Since we do not know whether the previous task has executed the offloading operation, the finish time of the predecessors needs to consider two situations, which should be defined as:

$$\underset{(v_k,v_i)\in\varepsilon}{Finish_T_k} = \begin{cases} Finish_T_k^{loc}, a_k = 0 \\ Finish_T_k^{trans}(1), a_k = 1, 2 \ldots, m(m \in M) \end{cases} \tag{4}$$

The finish time of task v_i executed locally is the sum of the execution time and the waiting time. The waiting time includes $Avail_T_i^{loc}$, and the finish time of the previous task $Finish_T_k$. Therefore, the local finish time can be defined as follows:

$$\begin{aligned} Finish_T_i^{loc} &= T_i^{loc} + T_i^{loc_wait} \\ &= T_i^{loc} + \max\left\{ Avail_T_i^{loc}, \underset{(v_k,v_i)\in\varepsilon}{Finish_T_k} \right\} \end{aligned} \tag{5}$$

Tasks are offloaded to the MEC server in three periods: the sending and uploading period, the MEC server execution period, and the download and receiving period. These will be discussed separately in the following paragraphs:

In the sending and uploading phase, the finish time is the sum of the upload time and the upload waiting time. The upload waiting time includes the finish time of the previous task and the earliest available uplink time. Therefore, the finish time of the upload can be defined as:

$$\begin{aligned} Finish_T_i^{up} &= T_i^{trans}(0) + T_i^{up_wait} \\ &= T_i^{trans}(0) + \max\left\{ Avail_T_i^{up}, \underset{(v_k,v_i)\in\varepsilon}{Finish_T_k} \right\} \end{aligned} \tag{6}$$

In the MEC server execution phase, the preceding period of a task includes two situations: The task has been completed in the sending and uploading phase, or the preceding task is also executed on the MEC server with the same number as the current task. Therefore, the finish time of the previous task can be defined as:

$$Finish_T_{pre} = \begin{cases} Finish_T_{i.m}^{ser} \\ Finish_T_k^{trans}(1), (v_k, v_i) \in \varepsilon \end{cases} \tag{7}$$

The task finish time of the MEC server numbered m is the sum of the task execution time and the waiting time. The waiting time includes the earliest task execution time of the

MEC server and the finish time of the previous item. Therefore, the finish time of task v_i on the MEC server numbered m can be defined as:

$$\begin{aligned} Finish_T_{i,m}^{ser} &= T_{i,m}^{ser} + T_{i,m}^{ser_wait} \\ &= T_{i,m}^{ser} + \max\left\{ Avail_T_{i,m}^{ser}, Finish_T_{pre} \right\} \end{aligned} \tag{8}$$

In the download and receive phase, the finish time shall be the sum of the download time and the waiting time for the download. The waiting time should include the finish time of the pre-task phase and the earliest available time of the download link. There is only one situation in the preceding phase of a task: the task has been completed on the MEC server. Therefore, the finish time of the download can be defined as:

$$\begin{aligned} Finish_T_{i}^{down} &= T_{i}^{trans}(1) + T_{i}^{down_wait} \\ &= T_{i}^{trans}(1) + \max\left\{ Avail_T_{i}^{down}, Finish_T_{i,m}^{ser} \right\} \end{aligned} \tag{9}$$

We should also consider the energy expenditure aspect. The total energy consumption cost includes computing energy consumption and transmission energy consumption. Since there is no transmission when the task is executed locally, only the computational cost needs to be considered. When offloading a task to the MEC host, there is only the transmission cost for the mobile device. Therefore, the total energy consumption cost of mobile devices can be calculated as:

$$\begin{aligned} E_{off}^{all} &= \sum_{v_i \in V, a_i = 0} E_i^{loc} + \sum_{v_i \in V, a_i = \{1,2\ldots,m\}} E_i^{trans} \\ &= \sum_{v_i \in V, a_i = 0} \kappa_u * \left(f_n^{loc}\right)^3 * T_i^{loc} + \sum_{v_i \in V, a_i = \{1,2\ldots,m\}} \left(P_{\text{send}} * T_i^{up} + P_{\text{rec}} * T_i^{down}\right) \end{aligned} \tag{10}$$

3.3. Problem Formulation

This section formalizes the offloading problem. Let $\xi = (V, E)$ denote DAG, where each vertex v_i in ξ represents a task ($v_i \in V$), and the directed edge $e(v_i, v_j) \in E$ represents the dependency between task v_i and task v_j. The next task can be started only when the current task is complete.

Any task in the DAG can choose whether to offload to the MEC host or run locally on UE. If task v_i is offloaded to MEC, the task execution is divided into the sending phase, execution phase, and receiving phase. In particular, the data transfer between the UE and the MEC host can be ignored for tasks executed locally. Let us assume that the set of tasks in the DAG is $N = \{1,2, \ldots, n\}$, and the set of MEC server is $M = \{1,2, \ldots, m\}$. Let a_i denote the offloading decision of task v_i. When $a_i = 0$, it means that v_i is scheduled to the local processor, and when $a_i = m$ ($m \in M$), it means v_i is offloaded to the MEC server numbered m.

SACDTO is designed to make offloading decisions with limited computing resources in multi-user and multi-MEC server scenarios. The offloading decision can shorten the completion time of all dependent tasks and optimize the system's energy consumption.

Objective function J is a linear weighting method for multi-objective optimization, which can be defined as:

$$J = \beta_1 \left(1 - \frac{\sum_{i=1}^{N} T_{off}}{T_{loc}^{all}}\right) + \beta_2 \left(1 - \frac{\sum_{i=1}^{N} E_{off}}{E_{loc}^{all}}\right) \tag{11}$$

where T_{off} and E_{off} denote the time and energy consumed to complete the task after the offloading decision is executed. T_{loc}^{all} and E_{loc}^{all} denote the sum of time and energy consumed when all tasks are running on the local device. The ratio shows the advantage of offloading decisions over local execution. Therefore, $1 - \frac{\sum_{i=1}^{N} E_{off}}{E_{loc}^{all}}$ is the ratio of reducing energy

consumption, and $(1 - \frac{\sum_{i=1}^{N} T_{off}}{T_{loc}^{all}})$ is the ratio of shortening completion time. β_1 and β_2 (β_1, $\beta_2 \in [0, 1]$, $\beta_1 + \beta_2 = 1$) represent the proportions of delay and power consumption, respectively. These two values can be adjusted according to user requirements. Our purpose is to maximize the objective function J:

$$\begin{aligned}
&\underset{A,F}{Max} J = \underset{A,F}{Max} [\beta_1(1 - \frac{\sum_{i=1}^{N} T_{off}}{T_{loc}^{all}}) + \beta_2(1 - \frac{\sum_{i=1}^{N} E_{off}}{E_{loc}^{all}})] \\
&A = [a_1, a_2, \ldots, a_i, \ldots, a_N] \\
&F = [f_1, f_2, \ldots, f_i, \ldots, f_N] \\
&f_i = \begin{cases} f_n^{local}, a_i = 0 \\ f_m^s, a_i \neq 0, m \in M \end{cases} \\
&s.t. \\
&C1 : a_i \in \{0, 1, \ldots, m\}, \forall m \in M, \forall i \in N \\
&C2 : f_i > 0, \forall i \in N \\
&C3 : \sum_{i \in N} f_i \leq C_m^{MEC}, \forall m \in M
\end{aligned} \tag{12}$$

where A is a task offloading decision vector and F is the computing resource allocation vector. Restriction $C1$ to $C2$ indicates that each task can only be offloaded to the local or one MEC server for calculation. $C3$ indicates that the computing resources allocated to task v_i by the MEC server numbered m do not exceed the total computing resources of the MEC server.

4. SACDTO Strategy

This section presents the implementation of the SACDTO strategy. Firstly, the SAC algorithm for a discrete action space is introduced. Secondly, we describe the offloading model based on the optimization problem. Finally, the implementation of the algorithm is presented.

4.1. SAC Algorithm for Discrete Actions

The SAC method is a DRL algorithm of offline maximum entropy. An online algorithm needs new samples every time to find the gradient, while an offline algorithm can use experience. Based on maximizing the expected reward, this algorithm uses the maximum entropy framework to explore and complete the task as randomly as possible. Compared with a deterministic policy (e.g., DDPG), this algorithm has stronger explorability, stability, and robustness.

In general, SAC is only suitable for continuous actions. However, this paper applies the SAC algorithm suitable for discrete actions, which is briefly introduced in the following section. One essential difference between these two SAC algorithms is that the output of the latter strategy, $\pi_\Phi(a_t, s_t)$, is probability rather than density.

Entropy is used to measure the randomness of random variables, and the entropy value of a policy $\pi_\Phi(a_t, s_t)$ can be calculated as:

$$H(\pi(\cdot|s_t)) = E[-\log \pi(\cdot|s_t)] \tag{13}$$

The goal of SAC is to maximize entropy and reward, which can be defined as:

$$\pi* = \underset{\pi}{\text{argmax}} E_{(s_t,a_t)\sim\rho_\pi}[\sum_t R(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] \tag{14}$$

Compared with the general reinforcement learning target, SAC adds entropy, and α denotes the temperature parameter, which indicates the emphasis on entropy. Therefore, this value can be used to adjust the proportion of system reward and entropy. In addition,

instead of setting this value as a hyperparameter, the temperature error is backpropagated and then adaptively adjusted through the network. The temperature target is defined as:

$$J(\alpha) = \pi_t(s_t)^T[-\alpha(\log(\pi_t(s_t)) + \overline{H})] \tag{15}$$

where $\overline{H}$ is a constant vector, which is equivalent to the hyperparameter of the target entropy.

The cumulative reward of entropy in state S_t is defined as a soft state value function. Since the action set is discrete, the expected value can be calculated directly without forming a Monte Carlo estimate:

$$V(s_t) := \pi(s_t)^T[Q(s_t) - \alpha \log(\pi(s_t)] \tag{16}$$

According to this value, the soft Q function of the offloading policy π can be obtained as:

$$Q(s_t) = r(s_t) + \gamma E_{s_{t+1} \sim \rho}[V(s_{t+1})] \tag{17}$$

To train soft Q function parameters, we try to minimize soft Bellman residuals:

$$J_Q(\theta) = E_{(s_t) \sim D}[\frac{1}{2}(Q_\theta(s_t) - \hat{Q}(s_t))^2] \tag{18}$$

where D represents the replay buffer and $\hat{Q}(s_t)$ is the target soft Q function. The gradient can be calculated as:

$$\hat{\nabla} J_Q(\theta) = \nabla Q_\theta(s_t)(Q_\theta(s_t) - (r^t + \gamma(Q_{\hat{\theta}}(s_{t+1}) - \alpha \log \pi_\Phi(s_{t+1})))) \tag{19}$$

In the SAC applicable to continuous actions, the re-parameterization technique is used to minimize policy loss $J_\pi(\Phi)$. However, this technique is not needed now. Therefore, the policy objective can be defined as:

$$J_\pi(\Phi) = E_{s_t \sim D}[\pi_t(s_t)^T[\alpha \log(\pi_\Phi(s_t)) - Q_\Phi(s_t)]] \tag{20}$$

4.2. The Task Offloading Model

The Markov prediction model is based on the Markov chain, which is a kind of memory-free discrete-time random process [29]. To solve the task offloading problem, we model the above optimization problem as MDP(S, A, R), where the state space, action space, and reward function of MDP are defined as follows.

4.2.1. State Space

The state space is defined as the combination of the vectors of DAG and the offloading decisions. Let $O_{1:i}$ denote a collection of offloading decisions from the first task to the current task and ξ denote the encoded DAG. In fact, ξ contains a vector group [X, Y, Z, U, V, K] formed by six vectors. Vector X represents the number of tasks in different states of progress, including the number of tasks in the upload link, the number of tasks in the download link, the number of offloading tasks, the number of local tasks, the number of tasks waiting to be processed, and the number of completed tasks; vector Y represents the total time to finish tasks with different schedules; vector Z represents the number of preceding and succeeding tasks; vector U represents the number of preceding tasks completed; vector V represents the sum of the preceding and subsequent task costs; vector K represents the cost of each schedule task to be processed next. Therefore, the state space can be expressed as:

$$S := \{s | s = (\xi, O_{1:i})\} \tag{21}$$

4.2.2. Action Space

The task can be executed on a local device or offloaded to an MEC host. If $a_i = 0$, the task Ti is executed locally. If a_i is any number except 0, the task is executed on the MEC host

with this value. Therefore, the action space can be defined as A: = {0,1,2, . . . , m}, where m represents the number of MEC servers.

4.2.3. Reward Function

The objective of this strategy is to maximize the system objective function J. QoS can be improved by reducing system delay and energy consumption. Since the reward function of each step is the increment in QoS, the reward function can be defined as:

$$R(s_i, a_i) = \beta_1[\frac{1}{N} - \frac{(\sum_{i=1}^{N} T_{off} - \sum_{i=1}^{N-1} T_{off})}{T_{loc}^{all}}] + \beta_2[\frac{1}{N} - \frac{(\sum_{i=1}^{N} E_{off} - \sum_{i=1}^{N-1} E_{off})}{E_{loc}^{all}}] \quad (22)$$

where N represents the total number of tasks and $\sum_{i=1}^{N} T_{off} - \sum_{i=1}^{N-1} T_{off}$ and $\sum_{i=1}^{N} E_{off} - \sum_{i=1}^{N-1} E_{off}$ represent the difference between the cost of the current offloading decision and the cost of the previous state. The larger the difference is, the better the current decision is, and the larger the reward value is.

4.3. SACDTO Implementation

The offloading process consists of the following three steps:

Step 1: The tasks of DAG are topologically sorted, and all task nodes are sorted according to the descending order of the rank value of the tasks, so as to obtain the task sequence, which can be defined as:

$$rank(v_i) = \begin{cases} T_tot_i, v_i \notin exit \\ \max_{(v_i, v_j) \in \varepsilon}(rank(v_j)) + T_tot_i, v_i \in exit \end{cases} \quad (23)$$

where T_tot_i = $Finish_T_i^{up}$ + $Finish_T_i^{up}$ + $Finish_T_i^{up}$ and *exit* represents the set of exit task nodes.

Step 2: According to the description of the state space in the offloading model in Section 4.2, the tasks are transformed into vector sequences, which are used as the input of the neural network. The output of the neural network is the probability of offloading.

Step 3: The offloading action with high probability is taken as the unloading decision a_i of task v_i, and the mobile device and the MEC host cooperate to complete task v_i according to the a_i.

The offloading decision of each task is made by SACDTO, and the detailed process of SACDTO is shown in Algorithm 1.

Algorithm 1: SACDTO

Input: Episode, environment, batch size, replay buffer size
Initialize the networks: $Q_{\theta 1} \rightarrow R^{|A|}$, $Q_{\theta 2} \rightarrow R^{|A|}$, $\pi_\phi \rightarrow [0, 1]^{|A|}$
Initialize the target network: $\hat{Q}_{\theta 1} \rightarrow R^{|A|}$, $\hat{Q}_{\theta 2} \rightarrow R^{|A|}$
Initialize the target network weights: $\hat{\theta}_1 \leftarrow \theta_1$, $\hat{\theta}_2 \leftarrow \theta_2$
Initialize the replay buffer: $D \leftarrow \varnothing$
for *each episode* **do**
    Obtain state S_t from environment;
    while *not done*:
        Allocate computing resources to each MEC server;
        Determine the current task to work on based on the priority list;
        $a_t \sim \pi_\phi(a_t \mid s_t)$;
        Update the remaining computing resources for each MEC server according to a_t;
        $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$;
        $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1}\}$;
        if *current episode % learning interval step==0* **then**

12	Sample a random minibatch of samples from D to calculate the target values;
13	Update the Q-function parameters $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J(\theta_i), i \in \{1,2\}$;
14	Update policy weights $\Phi \leftarrow \Phi - \lambda_\pi \hat{\nabla}_\Phi J_\pi(\Phi)$;
15	Update temperature $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$;
16	Update target network weights $\hat{Q}_i \leftarrow \tau Q_i + (1 - \tau)\hat{Q}_i, i \in \{1,2\}$;
Output: θ_1, θ_2, ϕ	

5. Performance Evaluation

This section gives the experimental results and performance evaluation of SACDTO. We first introduced the meaning of the different parameters in the DAG generator. Secondly, we set the simulation environment and hyperparameters, and give the convergence analysis of SACDTO an average reward. Finally, the performance of this algorithm is analyzed by comparing it with four benchmark algorithms.

5.1. Baseline Approaches

To verify the effectiveness and convergence of the proposed offloading strategy and evaluate the performance of this strategy under different parameters from multiple perspectives, we consider the following three computation offloading schemes for comparison.

- Proximal Policy Optimization-based Task Offloading (PPOTO): PPO is an improved algorithm of Policy Gradient, but still has the disadvantage of low sampling efficiency;
- Dueling Double Deep-Q Network-based Task Offloading (D3QNTO): Combines the advantages of Dueling DQN and Double DQN, and improves the training algorithm and model structure of DQN;
- Random Task Offloading (RTO): Computation tasks are offloaded randomly;
- All Local (ALOC): All computing tasks are executed locally.

5.2. Simulation and Results

In practice, applications can be modeled as DAGs with various topologies, but current real datasets contain information for only a very limited number of applications [30]. Therefore, if you want to use the scheme proposed in this paper to complete the offloading decision of applications, but these applications do not have relevant topological information in the real data set, you can use the synthetic DAG generator [31] to generate DAGs with various topologies to represent the dependencies between heterogeneous applications. The generator can adjust the aspect ratio (the aspect ratio determines the width of the DAG) and the Communication Calculation Ratio (CCR), and the changes in these two values will construct different topologies.

In the simulation experiment, according to the different distance between the terminal and the MEC host, there will be different channel gains and different transmission rates. Assuming that this is ignored, the default transmission rate is set to 1Mbps in the experiment. Set the CPU clock frequency f_i^{loc} of the local device to 1GHz and the CPU clock frequency ${f_m}^s$ of the MEC server to 8 GHz. The sending power P_{send} and receiving power P_{rec} are 1.26 W and 1.2 W, respectively. The DAG task configuration file size d_i is set to 5 KB to 50 KB, and the number of clock cycles required by a single task is set to 10^7 to 10^8 cycles/sec.

In each agent, the actor and the critics consist of an input layer, two hidden fully-connected layers, and an output layer. Both hidden fully connected layers are set to 256 neurons. We summarize the hyperparameters of SACDTO implementation in Table 3.

Figure 2 compares the convergence curves of the average rewards of SACDTO, PPOTO, D3QNTO, and RTO. Since the reward value function takes the local calculation strategy as the benchmark comparison, the convergence of all tasks executed locally can be ignored.

Table 3. Parameters for algorithm implementation.

Parameters	Value
Replay memory size	5000
Optimizer	Adam
Learning rate for Actor and Critic networks	0.001
Minibatch size	128
Discounted factor for reward	0.99
Delayed update factor	0.995
Initial value of temperature parameter	0.2
The learning rate of temperature parameter	0.001

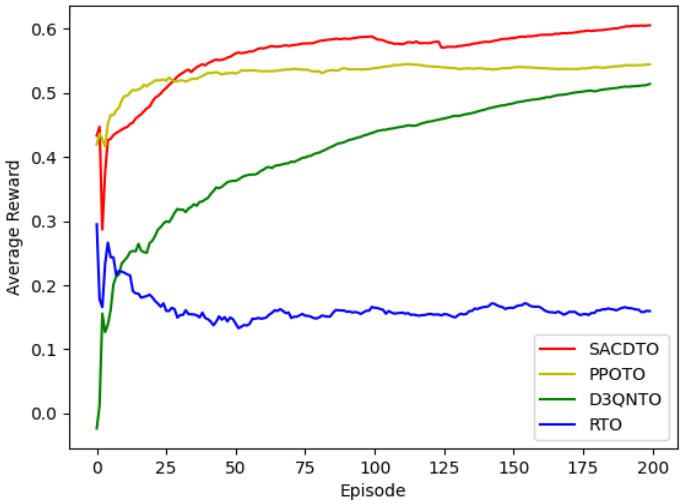


Figure 2. Convergence Analysis of Graph.

By default, we assume that the number of MEC servers is three and that we have 30 tasks to process. Obviously, as the number of episodes increases, the average reward value of the three schemes based on DRL increases until they converge. However, the convergence curve of RCO has been maintained at a low level, and there is no upward trend, which indicates that RTO cannot learn self-adaptive offloading action. D3QNTO converged gradually after the 150th round, and PPOTO converged at the 20th round. Although the PPOTO converged faster, the average reward of this strategy was less than D3QNTO by 5.74%. SACDTO strategy gradually converges after 125 rounds, and its result has 11.21% more reward than PPODTO, which is the best result among the above algorithms.

Since SAC can make use of all useful behaviors in each state and explore more useful behavior space through random strategy and maximum entropy target, the proposed SACDTO can obtain a higher average reward value compared with PPOTO and D3QNTO. This reflects that the SAC-based offloading scheme can learn more appropriate offloading strategies than PPO and D3QN.

Figure 3 shows the Time and Energy Reduced Scale (TERS) of the four offloading schemes when the number of tasks increases. TERS refers to the reduction ratio between the total cost calculated by the corresponding weight coefficient after unloading and the total cost consumed by the local task execution. Since it is a trade-off between delay and energy consumption, this value can also equal to QoS. Since TERS are also calculated against the local execution policy, ALOC can be ignored. By default, the aspect ratio of DAG is set to 0.45, the number of MEC servers is set to 3, and CCR is set to 0.5 (the smaller the value, the more intensive the computation). Compared with PPOTO, D3QNTO, and RTO, SACDTO has an average increment of 2.4%, 54.82%, and 79.28%, respectively, which indicates that SACDTO has the highest proportion of optimization compared with the local execution strategy. This is because with the increase in the number of tasks, the delay and

energy consumption generated by the local execution of all computing tasks will increase significantly, while the cost generated by the solution after the offloading decision increases relatively slowly. Therefore, the ratio of the two will be different, and the solution with good effect will have relatively higher TERS.

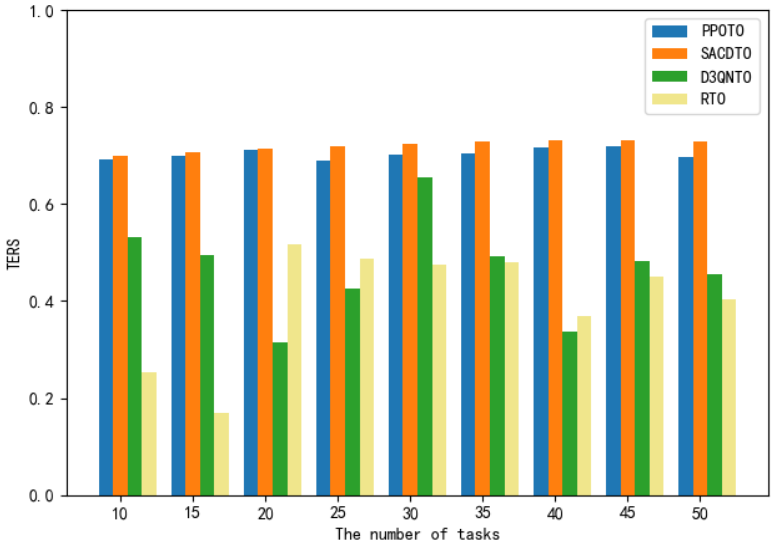


Figure 3. The Effect of Number of Devices on Delay.

Figures 4 and 5 respectively show the service latency and energy consumption of the five offloading schemes with an increasing number of tasks. Obviously, with the increase in the number of tasks, the total energy consumption and delay of these five strategies also increase. The latency and energy consumption of all locally executed computing tasks are increasing significantly, while the growth curves of service latency and energy consumption of the three DRL-based offloading strategies are relatively flat, which indicates that the above strategies have optimized the service latency and energy consumption of the system. In particular, SACDTO has an average reduction of 1.6%, 42.13%, 47.73%, and 62.19% in service latency and 78.49%, 6.7%, 89.12%, and 94.93% in energy consumption compared with PPOTO, D3QNTO, RTO, and ALOC, respectively. This shows that with the increase in the number of tasks, SACDTO learns a better offloading strategy, which can effectively reduce the delay and energy consumption.

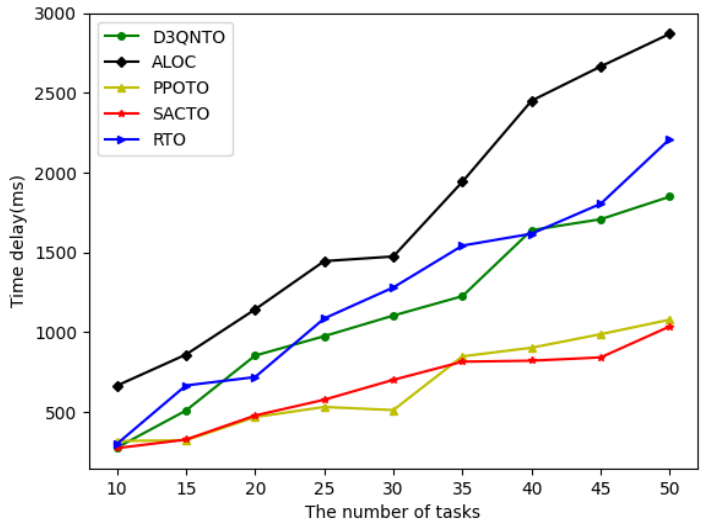


Figure 4. The Effect of Number of Tasks on Delay.

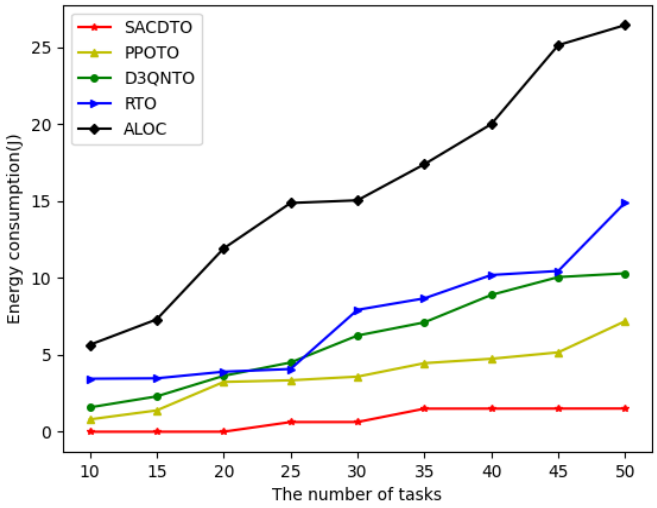


Figure 5. The Effect of Number of Tasks on Energy Consumption.

Figure 6 shows the relationship between CCR and TERS. CCR is the ratio between communication cost and computation cost. The smaller the value, the more intensive the computation task. In this experiment, the CCR is set between 0.2 and 0.6. We noticed that the more intensive the computation task, the higher the optimization ratio of SACDTO and PPOTO, while D3QNTO and RTO fluctuated greatly without obvious regularity. This is because the higher the cost of communication and the more intensive the task, the greater the latency and energy consumption of local computing. The better the offloading strategy, the more significant the cost reduction and, therefore, the greater the percentage reduction relative to the cost of local computing. In addition, the proposed SACDTO outperforms the other three strategies in the CCR of this value range, being 6.41%, 41.2% and 50.8% higher than PPOTO, D3QNTO, and RTO, respectively.

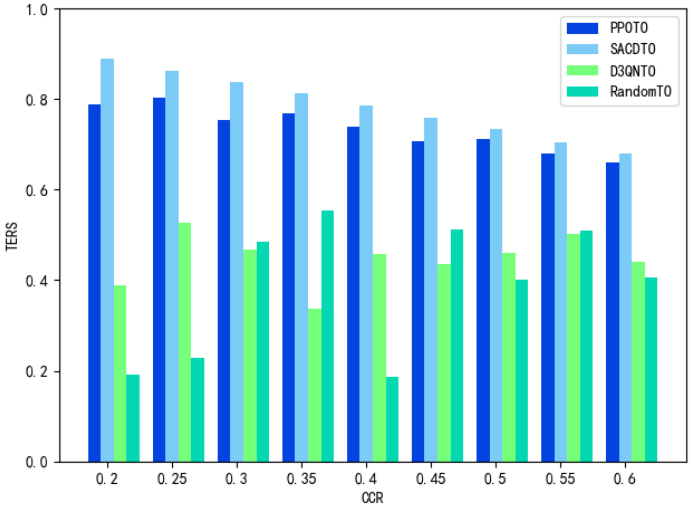


Figure 6. The Effect of CCR on TERS.

6. Conclusions and Future Work

In this paper, we investigate the offloading problem with dependent tasks, aiming to jointly optimize the delay and energy consumption in multi-server and multi-device scenarios and effectively improve the QoS. To cope with the dynamic scenario of MEC, we propose an edge–cloud collaborative MEC architecture, which models the optimization problem as an MDP. We use DAG to represent dependent tasks, embed DRL training in the MEC system, and use a neural network to approximate the policy function and value

function of MDP. This paper proposes an intelligent task offloading scheme SACDTO, which can offload each task to the corresponding MEC server through centralized control, greatly reducing the service delay and terminal energy consumption. Experimental results show that the algorithm converges quickly and stably, and has strong adaptability in different MEC scenarios. The optimization effect of the algorithm is better than ALOC, RTO, D3QNTO, and PPOTO. In the future, we plan to use multi-agent reinforcement learning to share the common resource pool. A UE can be regarded as an independent agent, and it should cooperate with other agents to realize the solution with the lowest cost and achieve the maximum QoS.

Author Contributions: Conceptualization, B.P.; Methodology, B.P.; Software, B.P.; Writing—original draft, B.P.; Writing—review & editing, T.L. and Y.C.; Funding acquisition, T.L.; Supervision, T.L.; Project administration, Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: Guangxi science and technology plan project of China: AD2029712; National Science Foundation of China: 61762010.

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not Applicable.

Data Availability Statement: Not Applicable.

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

References

1. Yan, M.; Li, W.; Chan, C.A.; Bian, S.; Chih-Lin, I.; Gygax, A.F. PECS: Towards personalized edge caching for future service-centric networks. *China Commun.* **2019**, *16*, 93–106. [CrossRef]
2. Wang, B.; Song, Y.; Cao, J.; Cui, X.; Zhang, L. Improving Task Scheduling with Parallelism Awareness in Heterogeneous Computational Environments. *Future Gener. Comput. Systems.* **2019**, *94*, 419–429. [CrossRef]
3. Liu, S.; Cheng, P.; Chen, Z.; Xiang, W.; Vucetic, B.; Li, Y. Contextual User-Centric Task Offloading for Mobile Edge Computing in Ultra-Dense Network. *IEEE Trans. Mob. Computing* **2022**. [CrossRef]
4. Fan, X.; Cui, T.; Cao, C.; Chen, Q.; Kwak, K.S. Minimum-Cost Offloading for Collaborative Task Execution of MEC-Assisted Platooning. *Sensors* **2019**, *19*, 847. [CrossRef] [PubMed]
5. Zhang, W.; Wen, Y. Energy-Efficient Task Execution for Application as a General Topology in Mobile Cloud Computing. *IEEE Trans. Cloud Comput.* **2015**, *6*, 708–719. [CrossRef]
6. Mao, N.; Chen, Y.; Guizani, M.; Lee, G.M. Graph Mapping Offloading Model Based On Deep Reinforcement Learning with Dependent Task. In Proceedings of the 2021 International Wireless Communications and Mobile Computing (IWCMC), Harbin, China, 28 June–2 July 2021; pp. 21–28.
7. Chen, J.; Yang, Y.; Wang, C.; Zhang, H.; Qiu, C.; Wang, X. Multitask Offloading Strategy Optimization Based on Directed Acyclic Graphs for Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 9367–9378. [CrossRef]
8. Leng, L.; Li, J.; Shi, H.; Zhu, Y.A. Graph convolutional network-based reinforcement learning for tasks offloading in multi-access edge computing. *Multimed Tools* **2021**, *80*, 29163–29175. [CrossRef]
9. Guan, X.; Lv, T.; Lin, Z.; Huang, P.; Zeng, J. D2D-Assisted Multi-User Cooperative Partial Offloading in MEC Based on Deep Reinforcement Learning. *Sensors* **2022**, *22*, 7004. [CrossRef]
10. Huynh, L.N.T.; Pham, Q.-V.; Pham, X.-Q.; Nguyen, T.D.T.; Hossain, M.D.; Huh, E.-N. Efficient Computation Offloading in Multi-Tier Multi-Access Edge Computing Systems: A Particle Swarm Optimization Approach. *Appl. Sci.* **2020**, *10*, 203. [CrossRef]
11. Ke, H.; Wang, H.; Sun, H. Multi-Agent Deep Reinforcement Learning-Based Partial Task Offloading and Resource Allocation in Edge Computing Environment. *Electronics* **2022**, *11*, 2394. [CrossRef]
12. Tang, M.; Wong, V.W.S. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Trans. Mob. Comput.* **2022**, *21*, 1985–1997. [CrossRef]
13. Hu, H.; Wu, D.; Zhou, F.; Jin, S.; Hu, R.Q. Dynamic Task Offloading in MEC-Enabled IoT Networks: A Hybrid DDPG-D3QN Approach. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6.
14. Liu, K.-H.; Hsu, Y.-H.; Lin, W.-N.; Liao, W. Fine-Grained Offloading for Multi-Access Edge Computing with Actor-Critic Federated Learning. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 3 March–1 April 2021; pp. 1–6.
15. Liu, J.; Lin, F.; Liu, K.; Zhao, Y.; Li, J. Research on Multi-Terminal's AC Offloading Scheme and Multi-Server's AC Selection Scheme in IoT. *Entropy* **2022**, *24*, 1357. [CrossRef]
16. Li, S.; Hu, S.; Du, Y. Deep Reinforcement Learning and Game Theory for Computation Offloading in Dynamic Edge Computing Markets. *IEEE Access* **2021**, *9*, 121456–121466. [CrossRef]

17. Wang, J.; Hu, J.; Min, G.; Zhan, W.; Ni, Q.; Georgalas, N. Computation Offloading in Multi-Access Edge Computing Using a Deep Sequential Model Based on Reinforcement Learning. *IEEE Commun. Mag.* **2019**, *57*, 64–69. [CrossRef]
18. Chen, X.; Liu, G. Federated Deep Reinforcement Learning-Based Task Offloading and Resource Allocation for Smart Cities in a Mobile Edge Network. *Sensors* **2022**, *22*, 4738. [CrossRef]
19. Dang, X.; Su, L.; Hao, Z.; Shang, X. Dynamic Offloading Method for Mobile Edge Computing of Internet of Vehicles Based on Multi-Vehicle Users and Multi-MEC Servers. *Electronics* **2022**, *11*, 2326. [CrossRef]
20. Sun, C.; Wu, X.; Li, X.; Fan, Q.; Wen, J.; Leung, V.C.M. Cooperative Computation Offloading for Multi-Access Edge Computing in 6G Mobile Networks via Soft Actor Critic. *IEEE Trans. Netw. Sci. Eng.* **2021**. [CrossRef]
21. Liu, P.; Ge, S.; Zhou, X.; Zhang, C.; Li, K. Soft Actor-Critic-Based DAG Tasks Offloading in Multi-Access Edge Computing with Inter-User Cooperation. In *Algorithms and Architectures for Parallel Processing*; Lai, Y., Wang, T., Jiang, M., Xu, G., Liang, W., Castiglione, A., Eds.; ICA3PP 2021; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2022; Volume 13157. [CrossRef]
22. Wang, B.; Wang, C.H.; Huang, W.W.; Song, Y.; Qin, X.Y. Security-aware task scheduling with deadline constraints on heterogeneous hybrid clouds. *J. Parallel Distrib. Comput.* **2021**, *153*, 15–28. [CrossRef]
23. Wang, B.; Cheng, J.; Cao, J.; Wang, C.; Huang, W. Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction. *Peer J. Comput. Sci.* **2022**, *8*, e893. [CrossRef]
24. He, W.; Gao, L.; Luo, J. A Multi-Layer Offloading Framework for Dependency-Aware Tasks in MEC. In Proceedings of the ICC 2021-IEEE International Conference on Communications, Montreal, QC, Canada, 14–18 June 2021; pp. 1–6.
25. Chen, L.; Wu, J.; Zhang, J.; Dai, H.-N.; Long, X.; Yao, M. Dependency-Aware Computation Offloading for Mobile Edge Computing with Edge-Cloud Cooperation. *IEEE Trans. Cloud Comput.* **2022**, *10*, 2451–2468. [CrossRef]
26. Long, J.; Luo, Y.; Zhu, X.; Luo, E.; Huang, M. Computation offloading through mobile vehicles in IoT-edge-cloud network. *J. Wirel. Com. Netw.* **2020**, *244*, 1–21. [CrossRef]
27. Dai, F.; Liu, G.; Mo, Q.; Xu, W.; Huang, B. Task offloading for vehicular edge computing with edge-cloud cooperation. *World Wide Web* **2022**, *25*, 1999–2017. [CrossRef]
28. Developing software for multi-access edge computing. *ETSI White Pap.* **2019**, *20*, 1–38.
29. Yan, M.; Li, S.; Chan, C.A.; Shen, Y.; Yu, Y. Mobility Prediction Using a Weighted Markov Model Based on Mobile User Classification. *Sensors* **2021**, *21*, 1740. [CrossRef] [PubMed]
30. 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]
31. Arabnejad, H.; Barbosa, J.G. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 682–694. [CrossRef]

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