electronics

MDPI

Article

Joint Optimization of Task Caching and Computation Offloading for Multiuser Multitasking in Mobile Edge Computing

Xintong Zhu, Zongpu Jia, Xiaoyan Pang and Shan Zhao *

School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo 454000, China; 212109020009@home.hpu.edu.cn (X.Z.)
* Correspondence: zhaoshan@hpu.edu.cn

Abstract: Mobile edge computing extends the capabilities of the cloud to the edge to meet the latency performance required by new types of applications. Task caching reduces network energy consumption by caching task applications and associated databases in advance on edge devices. However, determining an effective caching strategy is crucial since users generate numerous repetitive tasks, but edge devices and storage resources are limited. We aimed to address the problem of highly coupled decision variables in dynamic task caching and computational offloading for multiuser multitasking in mobile edge computing systems. This paper presents a joint computation and caching framework with the aim of minimizing delays and energy expenditure for mobile users and transforming the problem into a form of reinforcement learning. Based on this, an improved deep reinforcement learning algorithm, P-DDPG, is proposed to achieve efficient computation offloading and task caching decisions for mobile users. The algorithm integrates a deep and deterministic policy grading and a prioritized empirical replay mechanism to reduce system costs. The simulations show that the designed algorithm performs better in terms of task latencies and lower computing power consumption.

Keywords: deep reinforcement learning; task caching; computational offloading; mobile edge computing

Citation: Zhu, X.; Jia, Z.; Pang, X.; Zhao, S. Joint Optimization of Task Caching and Computation Offloading for Multiuser Multitasking in Mobile Edge Computing. *Electronics* **2024**, *13*, 389. https://doi.org/10.3390/electronics13020389

Academic Editors: Sahraoui Dhelim, Nyothiri Aung and Tao Zhu

Received: 12 December 2023
Revised: 11 January 2024
Accepted: 13 January 2024
Published: 17 January 2024

1. Introduction

The emergence of more compute-intensive mobile applications has put great pressure on current mobile network transmission [1]. Excessive latency and insufficient bandwidth are the problems that mobile edge computing (MEC), a novel computing paradigm, aims to address. It involves transferring computing and storage resources from mobile devices to the network edge, which is closer to the mobile user in terms of analyzing and generating knowledge. This approach overcomes the limitations of terminal capacity, lowering transmission latency and overhead and improving the quality of service (QoS) [2].

The success of computation offloading in MEC is heavily reliant on the efficient management of the limited computing, communication resources, and energy in the MEC system. Equipping MEC servers with base stations (BS) enables direct communication between devices to meet user demands for computing power, thereby reducing data transfer delays and battery power consumption [3]. However, MEC servers have limited computational resources, which still makes it difficult to meet the needs of users in terms of latency and energy consumption in communication environments with unstable transmission rates. Therefore, task caching techniques have been proposed. The limited cache space of MEC servers also affects users' offloading strategies and efficient processing of tasks. It is vital that the required services are properly cached on the MEC servers. Additionally, the mismatch between the computational and caching resources of MECs can further exacerbate the resource wastage at the edge, and collaboration is needed to make the best use of resources [4].

With the development of artificial intelligence, combining MEC networks with deep reinforcement learning (DRL) can work efficiently in non-linear and dynamic environments [5]. Through interactive learning from repeated experiences in a given environment, DRL can achieve long-term goals without prior knowledge and utilize its unique potential to dynamically select strategies in time-varying systems [6]. It solves optimization problems that are difficult to handle by traditional machine learning methods. And ultimately, it outputs the best long-term results. Deep deterministic policy gradients (DDPG) is a DRL algorithm for continuous action space [7]. It utilizes action-value functions to optimize policies, uses an auxiliary deterministic policy network for action selection, and improves the stability of the algorithm using empirical replay and goal networks. Compared to the conventional DRL algorithm, the DDPG algorithm utilizes a hybrid of policy-based and value-based techniques, and it performs better in dealing with the reinforcement learning problem in the continuous action space. By combining multiple algorithms, it can fully exploit the advantages of the deep neural network and is more suitable for solving the high-dimensional task offloading and allocation problem, especially when dealing with large-scale scenarios.

This study examines the challenge of jointly optimizing task caching and computation offloading in MEC systems with limited resources and multiple users and tasks. Initially, we formulate an optimization problem by developing a framework that integrates computation and caching. Subsequently, we focus on minimizing the execution delay and energy consumption of tasks performed by mobile users, taking into account both computation and caching aspects. Finally, we transform the formulated problem into a reinforcement learning problem and utilize the P-DDPG algorithm to attain the best decision regarding computation offloading and task caching. The key contributions of this study are as follows:

1. A new framework is designed for task caching and computation offloading in dynamic MEC environments by handling large-scale user requests under resource constraints. The combined optimization challenge of offloading computations and caching tasks is framed as a mixed-integer non-linear programming (MINLP) issue to reduce the system's average delay and power usage.
2. A P-DDPG algorithm is suggested for the combined optimization challenge of task caching and computational offloading, aiming to identify the most effective strategies for caching and offloading. Integrating the priority experience replay (PER) system disrupts the link between training experiences and enhances the accessibility of the experience replay buffer, thus boosting both the efficiency of training and the consistency of outcomes.

The subsequent sections of this document are structured in this manner: Section 2 outlines the associated research, Section 3 introduces the system model and problem formulation, Section 4 details the P-DDPG algorithm, Section 5 performs simulation experiments and analysis, and 6 provides the conclusion of this paper.

2. Related Work

Compute offloading aims to reduce compute, cache, and communication loads, effectively reducing latency and power consumption for compute-intensive tasks [8]. Mobile edge caching is a widely used internet technique that is a new method for alleviating network traffic by avoiding unnecessary, redundant processing during data transmission. Therefore, it improves the utilization of network resources and effectively reduces computational latency, energy consumption, and bandwidth costs.

The researchers in [9] investigated the problem of delay minimization in multiuser time division multiple access MEC offloading systems, which minimizes the weighted sum of all mobile device latency through the cooperation of cloud computing and MEC. The researchers in [10] encouraged a heuristic sub-optimal cache placement to implement an optimal caching policy using the branch bounding method to minimize communication and computation weights as well as energy consumption. In articles [11–13], data caching and computation offloading are jointly considered, and caching is employed to store

computation results of frequently used tasks, which helps to avoid the overheads associated with repetitive processing, provide better quality of service to users, and reduce data transmission. The authors in [14] proposed a method based on a long-term short-term memory (LSTM) network-based approach to predict the prevalence of tasks and, based on this approach, the joint optimality task unload decisions, compute resource allocation, and cache decisions to maximize long-term gains.

The study proposes task caching and computation offloading for static users in distributed MEC networks. However, static caching policies cannot maintain the high reusability of cached data as users' demands for computation tasks change dynamically over time [15]. Several studies have been conducted using DRL in MEC to solve complex optimization problems and delay-sensitive problems in time-varying systems. This involves addressing computational load and task caching and combines reinforcement learning with deep convolutional neural networks to improve algorithmic performance through caching and offloading tasks. DQN is a deep Q-network algorithm that employs a deep neural network to estimate the value function of an action [16]. Through continuous training and optimization of the network, an optimal action strategy can be derived, resulting in improved cumulative returns. The researchers in [17] used the DQN algorithm to provide an in-depth consideration of the system latency and computational cost of the edge server. The decision is made to minimize system latency and ensure that the total computational cost remains within the vehicle's budget. The authors in [18] recommended a multi-intelligence DQN algorithm based on predictive popularity. The algorithm optimizes computational offloading, resource allocation, and cache placement by reducing the overall delay in offloading and network resource utilization. The authors in [19] considered the computational offloading issues into optimization issues in terms of effort and time, and the optimal cost strategy is found using deep Q-network algorithms in reinforcement learning. Traditional DRL-based mechanisms can accomplish offloading, but their performance during training is not satisfactory. DDQN is an augmented learning algorithm that efficiently approximates the Q-value function using deep neural networks [20]. It requires no prior knowledge of network dynamics and can be used instead of DQN to learn optimal computational offload policies. The training speed is increased, and good performance is achieved by avoiding overestimation of the algorithm. The authors in [21] explored the combined enhancement of service caching, resource distribution, and partial computation offloading in MEC systems, utilizing the DDQN method amidst demand unpredictability. The goal is to reduce the aggregate weighted cost of delay and energy expenses. The authors in [22] introduced a DDQN offloading algorithm aimed at improving the task's offloading decision-making process. The algorithm is designed to reduce the overall delay and waiting period for mobile vehicle tasks, enhancing the management of computationally demanding tasks for dynamic and effective offloading choices. In their work, the authors in [23] suggested a DDPG-based algorithm, taking into account computation offloading, service caching, and resource distribution, to reduce task burden on mobile users by jointly exploiting the ES's computational and caching capabilities. Through the incorporation of the DDPG algorithm, the researchers in [7] developed a combined computational and caching system aimed at reducing energy expenses in telematics situations needing mobile network assistance from network providers.

However, there is less research on caching and offloading of multitasks in dynamic edge environments and changing resource states. Therefore, this paper looks at efficiently solving the online offloading issue for multitasks in time-varying systems with joint task caching to reduce redundant computations and transmissions between mobile equipment and MEC servers. During task execution, multiple mobile devices can share the computation and results to achieve the optimal policy. The proposed P-DDPG algorithm uses the accumulated training experience to guide the retraining, which greatly accelerates the learning process of executing different offloading tasks, enhances the effectiveness of the offloading policy, and reduces the overhead of the decision-making process.

3. System Model and Problem Formulation

The text is a description of a multiuser, multitask MEC system, which is a cache-and-offload model. Definitions of the symbols used in the system are given in Table 1.

Table 1. Symbol definition.

Symbol	Definition
M	Number of users.
K	Number of tasks.
$\Upsilon_{i,t}$	Upstream transmission rate of users.
p_i	Transmission power of users.
P_t	Total data upstream rate.
B	Wireless transmission bandwidth.
f_i^{loc}	The number of CPU cycles per user time.
$v_{i,j}$	The amount of data downloaded by tasks for offloading.
$b_{i,j}$	CPU cycles needed for task execution.
$\tau_{i,j}$	Task deadline.
$d_{i,j}$	Task offloading decision.
$c_{i,j}$	Caching decision of tasks.
F_{mo}	Total computing resources of MEC.
F_{co}	Cache size of MEC.
σ^2	Background noise power.
ς_n	Energy coefficient of mobile devices.
h^2	Channel gain.
α_i^E	Weight of energy consumption.
β_i^T	Weight of time consumption.
$T_{i,j}^{loc}$	Execution time of tasks offloaded locally.
$T_{i,j}^{e}$	Execution time of tasks offloaded to MEC.
$E_{i,j}^{loc}$	Energy consumption of tasks offloaded locally.
$E_{i,j}^{edge}$	Energy consumption of tasks executed on MEC.
$E_{i,j}$	Total energy consumption for task completion.
$T_{i,j}$	Total delay for task completion.
$S_{i,j}$	Total computing cost for task completion.

3.1. Network Model

Figure 1 shows the MEC system model, which comprises a MEC server and M mobile users, each containing k tasks. Users have a wireless connection to the MEC, while the MEC server has a fiber optic connection to the cloud server. The base station is directly connected to the MEC server to provide computing services and store data for user devices such as mobile phones and virtual realities (VRs) devices [24]. The specific system operation is as follows: the set of users is $M = \{1, 2, \ldots, M\}$, and the set of tasks is $K = \{1, 2, \ldots, \mathrm{K}\}$. The system can be completed in a sequence of time slots of the same length τ,$\tau = \{1, 2, \ldots, t\}$. At the start of each period, each mobile user starts request tasks, and each user's need for a computational task can be expressed in the form of a tuple $\{v_{i,j}, b_{i,j}, \tau_{i,j}\}$, where $v_{i,j}$ indicates whether the user has completed the offloading task, $b_{i,j}$ indicates the number of CPU cycles that can be executed per unit time, and $\tau_{i,j}$ denotes the deadline. Each task must satisfy the implementation delay and power consumption requirements and must be completed before the end of the current time slot. Each user completes the task through distributed computing, i.e., each user can process in parallel on multiple devices at the

same time; some tasks can be processed locally, and others are computed and processed by MECs.

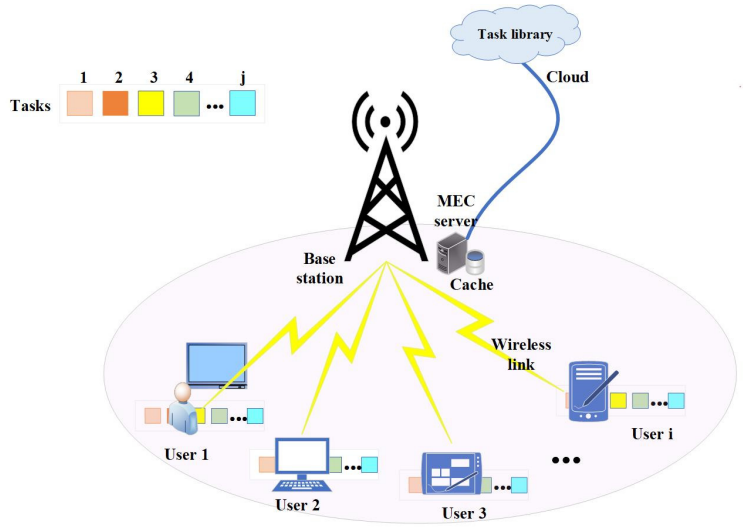

Figure 1. System model of MEC network.

3.2. Communication Model

Assuming that the user and the edge device are communicating via orthogonal frequency division multiple access (OFDMA) [25], the total bandwidth required by all the users is B MHz. To maximize the utilization of the wireless bandwidth, orthogonal bandwidths B are allocated to each user based on the effect of the bandwidth on the system overhead. This prevents interference between two users on orthogonal channels. Thus, the communication rate between users and the MEC is given by Shannon's formula:

$$\Upsilon_{i,t} = \mathrm{B}\log_2\left(1 + \frac{p_i h_{i,t}^2}{\sigma^2}\right), \tag{1}$$

3.3. Computation Model

In order to make sure that the results are complete, it is assumed that user-generated tasks can be computed sequentially, either locally or solely by the MEC server. Let a variable $d_{i,j}$ denote the computational offloading decision variable for user i, $d_{i,j} \in \{0,1\}$. Specifically, if $d_{i,j} = 0$, the ith user chooses local execution; if $d_{i,j} = 1$, the ith user offloads the task to the MEC server over the wireless connection. The computational offloading of all users is represented by the binary variable $d = \{d_{1,1}, d_{1,2}, d_{1,3}, \ldots, d_{i,j}\}$.

1. Local computation model;

During local execution, user M executes the computational task on their own CPU. f_i^{loc} denotes the computational power of the ith user in CPU cycles per second. Then, the execution delay depends on each one's computational power, so the local computation time of the task is:

$$\mathrm{T}_{i,j}^{loc} = \frac{b_{i,j}}{f_i^{loc}}, \tag{2}$$

Assuming that all mobile equipment has the same computing capacity, the energy consumption required for the computation process is expressed as:

$$E_{i,j}^{loc} = \varsigma_n b_{i,j}\left(f_i^{loc}\right)^2, \tag{3}$$

where the energy coefficient $\varsigma_n = 10^{-27}$[26,27] of the mobile device is determined by the slice structure.

2. MEC offloading model;

The MEC server mainly performs three steps: data transmission, task processing, and receiving the result to execute the task. When $d_{i,j} = 1$, the ith user must first access the BS in the area via the wireless network and transmit the data to the MEC server. After this, the MEC server assigns computing resources to execute the task and returns the task execution result to the mobile. Therefore, the transmission and task execution delay of mobile user i in time slot t is as follows:

$$\mathrm{T}_{i,j}^{off} = \frac{v_{i,j}}{\Upsilon_{i,t}}, \tag{4}$$

$$\mathrm{T}_{i,j}^{exec} = \frac{b_{i,j}}{f_i^{edge}}, \tag{5}$$

The full delay and energy consumed by the mobile user i when trying to perform a task on the MEC server can be expressed as follows:

$$\mathrm{T}_{i,j}^{edge} = \frac{v_{i,j}}{\Upsilon_{i,t}} + \frac{b_{i,j}}{f_i^{edge}}, \tag{6}$$

$$\mathrm{E}_{i,j}^{edge} = E_{i,j}^{off} = \frac{p_i v_{i,j}}{\Upsilon_{i,j}}, \tag{7}$$

The computing power of the MEC server is allocated to the user in proportion to the computing power of the mobile device. Meanwhile, because the size of the results is much smaller than the size of the input data, the delay in the transmission of the results is negligible [18,21].

3.4. Caching Model

In a multiuser MEC scenario, the same type of mobile users will repeatedly execute the same tasks within a short period, so task caching techniques are needed to enhance the service efficiency of computing devices. Equipping the MEC server with a limited cache space to cache the raw counts and codes of all applications can lead to better data sharing and computation during task execution and reduce task latency and user energy consumption.

The caching mechanism during task computation is as follows: each mobile user needs to offload and complete the task on the edge server; the MEC server first gives all the relevant information of the task, such as relevant code, task request, and computation information. Then, the MEC server can decide which task to cache; if the task is not cached, the application and relevant code will be offloaded; if the task is cached, the programmer will execute the computation task and return the result. Finally, the optimal caching policy will be found for the task to be cached.

Let the binary variable $c_{i,j} \in \{0, 1\}$ represent the caching decision of mobile user i for task j. $c_{i,j} = 0$ indicates that task j of user i is not cached, while $c_{i,j} = 1$ indicates that task j of user i has been cached by the MEC server. Thus, the caching decision profile $C = \{c_{1,1}, c_{1,2}, \ldots, c_{i,j}\}$ can be obtained. If the input data of task j is already cached in the MEC server when task j is computed, the corresponding data can be used in the next time slot, and the task can be executed entirely on the server. In this way, cached tasks do not need to be unloaded, there will be no latency in the system, there will be no energy cost incurred by the user in transferring task-specific data to the server, and the user's experience will be significantly improved. However, since the MEC server has limited

storage capacity and cannot cache all tasks, the caching decision variables for tasks must satisfy the following constraints.

$$\sum_{i=1}^{N}\sum_{j=1}^{M} c_{i,j}v_{i,j} \leq F_{co}\ , \forall \mathrm{t} \in \tau\ , \tag{8}$$

3.5. Problem Formulation

Since tasks that need to be offloaded are delay-sensitive and energy-consuming, the aim of this research is to develop optimal caching policies and offloading policies for all users' tasks that minimize the weighted and users' delay and energy consumption. The computational delay and energy consumption for completing task j is expressed as:

$$T_{i,j} = c_{i,j}T_{i,j}^{exec} + (1-c_{i,j})[(1-d_{i,j})T_{i,j}^{loc} + d_{i,j}T_{i,j}^{edge}]\ , \tag{9}$$

$$E_{i,j} = (1-c_{i,j})[(1-d_{i,j})E_{i,j}^{loc} + d_{i,j}E_{i,j}^{edge}]\ , \tag{10}$$

Finally, combining the offloading and caching decision, the cost of the t time slot MEC system is obtained as:

$$\mathrm{S_{i,j}} = \alpha_i^E E_{i,j} + \beta_i^t T_{i,j}, \tag{11}$$

where the parameters α_i^E and β_i^T represent the weights between energy and delay consumption, and their values reflect how much the system favors energy and delay. Thus, the problem can be formulated as:

$$P: \min_{D,C} \sum_{i=1}^{M}\sum_{j=1}^{K} \alpha_i^E E_{i,j} + \beta_i^t T_{i,j}, \tag{12a}$$

$$s.t. \quad \mathcal{C}1: \sum_{i=1}^{M}\sum_{j=1}^{K} d_{i,j}\gamma_i \leq P_t, \tag{12b}$$

$$\mathcal{C}2: \sum_{i=1}^{M}\sum_{j=1}^{K} d_{i,j}f_i^{edge} \leq F_{mo}\ , \tag{12c}$$

$$\mathcal{C}3: \sum_{i=1}^{M}\sum_{j=1}^{K} c_{i,j}v_{i,j} \leq F_{co}\ , \tag{12d}$$

$$\mathcal{C}4: T_{i,j} \leq \tau_{i,j}, \tag{12e}$$

$$\mathcal{C}5: d_{i,j} \leq c_{i,j}, \tag{12f}$$

$$\mathcal{C}6: d_{i,j} \in \{0,1\}, \tag{12g}$$

$$\mathcal{C}7: c_{i,j} \in \{0,1\}, \tag{12h}$$

Constraint C1 ensures that the allocated resources for all users do not exceed the available bandwidth. Constraint C2 states that the computed tasks must not exceed the maximum capacity of the MEC server's computational resources. Constraint C3 is a response to Equation (8), and it indicates that the full number of cached tasks must not exceed the storage capacity of the MEC server. Constraint C4 requires each user to complete their task within the current time slot. Constraint C5 prevents local computations from being cached on the MEC server. Constraints C6 and C7 ensure that decisions to offload and cache tasks are binary variables.

To find the minimum objective function P, it is important to make the optimal caching decisions C and offloading decisions D for each time slot. Since the values of C and D are binary, the action and state spaces of different user decisions will grow exponentially with the increase in the number of feasible sequences and tasks. This problem is non-convex and is NP-hard [5]. Furthermore, a noteworthy correlation exists between the task caching

strategy and the computational offloading strategy, which poses a challenge in achieving an optimal solution to the given problem. In this case, it is challenging for traditional decision-making algorithms, such as decentralized and heuristic algorithms, to find the optimization objective in a specific time frame. A feasible approach to efficiently solve the caching and offloading decision-making problem is to design a DRL method.

4. P-DDPG Algorithm

The optimization question is modeled as a Markov decision process (MDP) [28], which defines the state space, action space, and rewards. The proposed algorithm, P-DDPG, aims to minimize latency and energy consumption by finding the optimal caching and offloading scheme.

4.1. Formulation of the Problem with DRL

4.1.1. State Space

The system state space $s(t)$ is the basis for intelligence agents to make decisions and assess long-term benefits, reflecting the intelligence agent's perception of the environment, including task data size, cache capacity, and computation. The state $s(t)$ of the system in time slot t is defined as follows:

$$s(t) = \{X_{1,1}(t), X_{1,2}(t), \ldots, X_{i,j}(t), \mathrm{I}_{1,1}(t), I_{1,2}(t), \ldots, I_{i,j}(t), Y_{1,1}(t), Y_{1,2}(t), \ldots, Y_{i,j}(t)\}, \quad (13)$$

where $X(t)$ denotes the amount of computation for all tasks, $\mathrm{I}(t)$ denotes the data size of the task, and $Y(t)$ denotes the MEC cache capacity required by the task.

4.1.2. Action Space

For the observed environmental states, the intelligent body will generate a better caching strategy for the task and an offload policy for computer tasks. Thus, the problem to be solved in the action space is where to offload the task and whether to cache the task or not. The action $a(t)$ in time slot t is denoted as:

$$a(t) = \{d_{1,1}(t), d_{1,2}(t), \ldots, d_{i,j}(t), c_{1,1}(t), c_{1,2}(t), \ldots, c_{i,j}(t)\}, \quad (14)$$

where $D = \{d_{1,1}, d_{1,2}, \ldots, d_{i,j}\}$ denotes the offloading decision and $C = \{c_{1,1}, c_{1,2}, \ldots, c_{i,j}\}$ denotes the caching decision.

4.1.3. Reward

The reward function $\mathrm{R}(s(t), a(t))$ is a feedback identification for the intelligence to act and react to changes in the environment, and in DRL, it usually maximizes the long-term benefits of the MEC system. Therefore, the reward is determined by calculating the objective function P. Throughout the process, the intelligent body selects actions and continuously updates its strategy with higher rewards by calculating the expected rewards to reduce the cost of long-term actions. Negative values maximized according to the objective function P are used as rewards, and when rewards are accumulated to a maximum value, the task cost is reduced. Therefore, for a given state and action, the following equation describes the relationship between the reward and P's objective function:

$$r(t) = R(s(t), a(t)) = -\frac{1}{N}\sum_{i=1}^{M}\sum_{j=1}^{K}\alpha_i^E E_{i,j} + \beta_i^t T_{i,j}, \quad (15)$$

4.2. P-DDPG Algorithm Design

MEC is a dynamic system with high real-time requirements, and the DDPG algorithm adopts a combination of policy-based and value-based approaches, which obtains good results in solving continuous problems and has a faster convergence speed and higher performance. However, its efficiency needs to be further improved as the use of the experience replay mechanism is randomly selected from the experience base, and an

appropriate strategy is not selected for the best experience to learn. Therefore, the P-DDPG algorithm seeks optimal caching and offloading by introducing the PER mechanism.

Figure 2 displays the framework of the P-DDPG algorithm. The framework mainly includes the environment, actor network, critic network, and experience pool modules. The intelligent body obtains experience from the environment and stores it in the pool of experience for learning in the future. In the training process, the PER mechanism breaks the dependence on the training experience, enhances the influence of past relevant experience on the current state decision, and prevents the neural network from premature fitting. Additionally, adding random Gaussian distributed behavioral noise (Ornstein-Uhlenbeck process) [29] output to the actor network is a mean reversion process that makes the network more exploratory and prevents local optima.

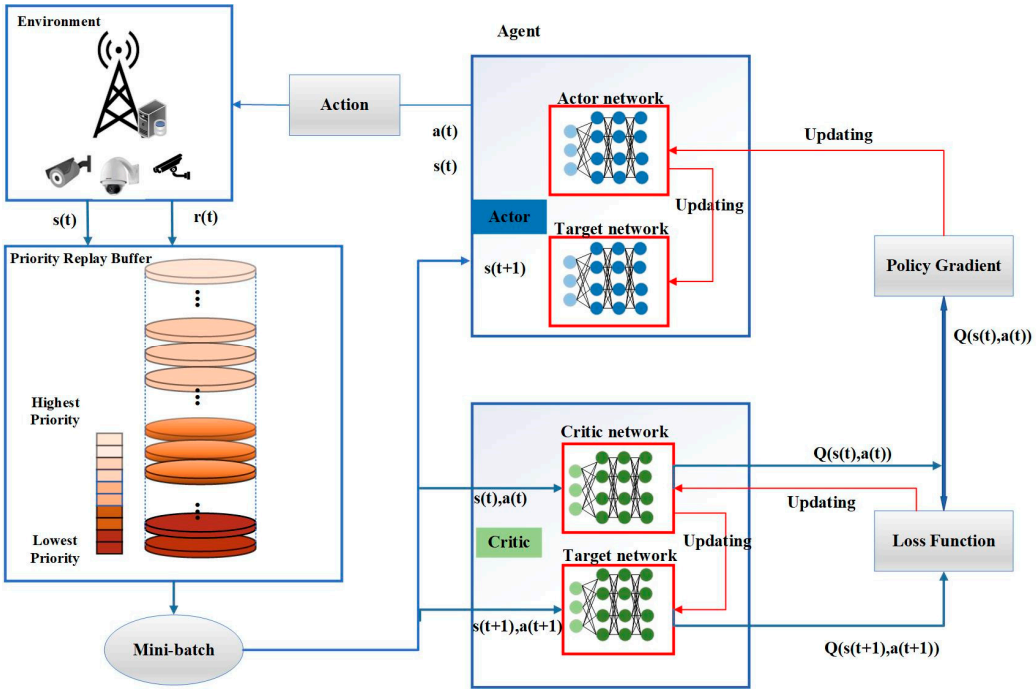

Figure 2. P-DDPG algorithm framework.

Specifically, the intelligent body obtains the current state $s(t)$, completes the action $a(t) = \mu(s(t); \theta^{\mu}) + N_t$, and computes the reward $r(t)$ according to Equation (15), after which it accesses the next state $s(t+1)$ through parameters θ^{μ} and random noise N_t of the current policy network. In the experience buffer D, the intelligent body stores the strategy μ, the value function $Q^{\mu}(s(t), a(t))$, and the transition information $(s(t), a(t), r(t), s(t+1))$. Then, N transition messages are randomly selected in D, and their parameters are trained. An estimate of the value function of the action is obtained from Bellman's equation describing the definition of the value function:

$$Q^{\mu}(s(t), a(t)) = \mathrm{E}\Big[r(s(t), a(t)) + \gamma Q^{\mu}\Big(s(t+1); \theta^{Q}\Big)\Big], \tag{16}$$

The PER mechanism is used in D to construct the experience correlation function and select the most appropriate learning experience at different time steps. The core principle of the PER mechanism is to measure the importance of each state transition. The correlation of TD error can be taken as an evaluation standard for the priority of the experience. It presents the measurements that the intelligent body has learned from past experiences to update the estimated action value function $Q^{\mu}(s(t), a(t))$. The TD error can be used as an

evaluation criterion for experience priority. The TD error is, therefore, calculated in the following way:

$$\delta_i = r(s(t), a(t)) + \gamma Q'\left(s(t+1), \mu'\left(s(t+1); \theta^{\mu'}\right); \theta^{Q'}\right) - Q(s(t), a(t); \theta^{Q}), \tag{17}$$

where δ_i specifies the degree of preferential learning, and the higher the δ_i, the stronger the correlation is, so there is still much to be done to improve prediction accuracy. At each step in the learning process, the TD error is calculated for all samples, and the sampling probability is calculated based on their experience:

$$P(i) = \frac{H_i^{\psi_1}}{\sum_k H_k^{\psi_1}}, \tag{18}$$

where the parameter ψ_1 controls the degree of priority ordering, and when ψ_1 = 0, it indicates uniform sampling.

A low TD error has a probability of repetition, and the sampling probability acts as a near-random factor in the chosen empirical process to prevent premature adaptation of the neural network. Excessive replay of the TD error with higher priority can lead to a decrease in sample diversity and a change in the frequency of state accesses, potentially causing the neural network to oscillate or diverge during training. Therefore, importance sampling is employed to correct the weights so that the true results in the relevant states can be obtained while acting as a high value. The importance sampling is calculated as follows:

$$\omega_i = \left(\frac{1}{D} \cdot \frac{1}{p(i)}\right)^{\psi_2}, \tag{19}$$

When ψ_2 = 1, it compensates for non-uniform probabilities. The use of importance sampling reduces the error and order of magnitude of the strategy gradient and uses more efficient experiences for learning to improve efficiency.

In each iteration, the target actor and the target critic network are combined using Calculation (21), the computed output is given to the main critic network, and the parameters of the online critic neural network are updated by minimizing the mean squared error loss function to minimize the loss function $L\left(\theta^{Q}\right)$, which can be expressed as:

$$L\left(\theta^{Q}\right) = \frac{1}{N}\left(Y(t) - Q\left(s(t), a(t); \theta^{Q}\right)\right)^2, \tag{20}$$

$$Y(t) = r(t) + \gamma Q'\left(s(t+1), \mu'\left(s(t+1); \theta^{\mu'}\right); \theta^{Q'}\right), \tag{21}$$

where N is the number of experiences used for learning. After that, the actor network is updated using the sampling strategy gradient:

$$\nabla_{\theta^{\mu}} J(\mu) = \frac{1}{N} \nabla_a Q\left(s(t), a(t); \theta^{Q}\right) \cdot \nabla_{\theta^{\mu}} \mu(s(t); \theta^{\mu}), \tag{22}$$

Finally, in order to improve the stability of the training, a soft update method is used instead of replicating the parameters θ^{Q} and θ^{μ} to update the target network parameters.

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \tag{23}$$

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{24}$$

where $\tau \in [0, 1]$ is used to determine the extent of updating.

Algorithm 1 displays the pseudocode of the P-DDPG algorithm. Firstly, the initialization environment is simulated, and in each time slot, the agent generates new states and rewards according to the current state $s(t)$ and the current policy by generating the corresponding actions. Then, the intelligent body transmits the tuple $(s(t), a(t), r(t), s(t+1))$

based on the samples and stores it in the priority experience playback buffer. A small batch I is formed based on the most relevant experiences sampled from the priority experience pool D. Subsequently, the selection probabilities of these experiences are updated based on the TD error, the priorities of the experiences are updated, and finally, the network parameters are updated using these experiences.

Algorithm 1: P-DDPG algorithm

Initialize the priority experience playback buffer D, the minimum batch size N, the TD error sample size N_{td}, the number of training times K, the iteration time slot T, and the weight control parameters ψ_1 and ψ_2.
Randomly initialize the weights θ^Q and θ^μ, discount factor λ, and update factor T of the main Q network and target Q network.
Initialize the weight parameters $\theta^{\mu'} \leftarrow \theta^\mu$, $\theta^{Q'} \leftarrow \theta^Q$ of the target network.
1: Initialize the main and target networks.
2: For $episode = 1, 2, \cdots, M$ do.
3: Randomly generate and receive initialized observation states $s(t)$.
4: Add random noise N_t for action exploration.
5: For $t = 1, 2, \cdots, T$ do.
6: The agent observes the state $s(t)$ and selects actions according to the current strategy and noise N_t.
7: Calculate the instant reward $r(t)$, and obtain the next immediate state after executing the action.
8: Based on the sampling probability in Equation (18), store $(s(t), a(t), r(t), s(t+1))$ and add it to the priority experience playback buffer D.
9: Form a small batch I by sampling the most relevant experiences from the priority experience pool D.
10: Calculate the weight of the importance sample $\omega_i = (\frac{1}{D} \cdot \frac{1}{p(i)})^{\psi_2}$.
11: Calculate the target Q value $Y(t) = r(t) + \gamma Q'\left(s(t+1), \mu'\left(s(t+1); \theta^{\mu'}\right); \theta^{Q'}\right)$.
12: Calculate the sampling probability of the TD error update experience from Equation (17).
13: Update the priority of the experience $P_i \leftarrow |\delta_i|$.
14: Update the θ^Q of the critical network according to the loss function Equation (20).
15: Update the weights of the actor network parameters θ^μ according to the strategy gradient strategy Equation (22).
16: Update the target network parameters according to Equations (23) and (24).
17: End for
18: End for

Four neural networks contribute significantly to the time complexity of Algorithm 1. The training algorithm can be considered to consist mainly of an actor network, a critic network, and a priority experience replay buffer. The actor and critic networks of each agent are processed by two DNNs to form the goal and evaluation networks. Consequently, the time complexity can be described as shown below [30].

$$2 \times \sum_{\iota=0}^{I} n_{actor,\iota} n_{actor,\iota+1} + 2 \times \sum_{l=0}^{L} n_{critic,l} n_{critic,l+1} = O\left(\sum_{\iota=0}^{I} n_{actor,\iota} n_{actor,\iota+1} + \sum_{l=0}^{L} n_{critic,l} n_{critic,l+1}\right), \quad (25)$$

where, I and L represent the number of fully connected layers for actor and critic DNN networks, respectively, $n_{actor,\iota}$ and $n_{critic,l}$ denote the number of units corresponding to the fully connected layers.

5. Performance Evaluation

5.1. Experimental Setup

A MEC system consisting of six cameras distributed at different angles and positions is considered to build a simulation environment for task caching and computational offloading. The system is linked to the BS via a wireless channel, and it has both computational and storage capabilities. The main task of the camera is to capture the video information and pre-process the data, analyze and track the video using AI techniques and models, identify the targets, and generate reports. The more information and data are collected, the more computation and processing are required. The cameras are randomly distributed in a 200 m × 200 m radius, and the BS has a coverage radius of $d_m = 200$ m in the center of the area [11]. The offloaded data of each mobile user are uniformly distributed within (0,30) MB, and considering the heterogeneous computing power of the cameras, the CPU frequency of each user is randomly assigned as {0.8, 0.9, . . ., 1.5} GHz, and the number of

cycles of the task execution CPU is $b_{i,j} = 500$ Cycles/bit [31]. The cache size of the MEC server is $F_{co} = 600$ MB, the computational resources of the MEC server are $F_{mo} = 30$ MHz, and the available channel bandwidth is 20 MHz [23,32,33]. The experiments were developed using TensorFlow 2.0, a Python 3.7 simulator for the AMD Ryzen processor. In the P-DDPG algorithm, the same four-layer fully connected neural network is used as the critic and actor networks. The first and second hidden layers use 256 and 128 neurons, respectively, and the learning rate for the critic network and actor network is 0.0001 and 0.001, respectively [34,35]. During the network training, the experience playback buffer size was set to 50,000, the minimum number of batch sampled experiences was 128, the soft update was 0.001, and the deduction factor was 0.99 [36]. Table 2 shows the specific settings quoted above to simulate the real environment.

Table 2. Simulation Parameters.

Parameter	Value
M	5
K	3
f_i	{0.8, 0.9, . . ., 1.5} GHz
$b_{i,j}$	500 Cycles/bit
$v_{i,j}$	(0,40) MB
F_{mo}	30 GHz
B	20 MHz
d_m	200 m
Pi	100 mW
σ^2	−100 dBm
N	128
a^Q	0.0001
a^μ	0.001
γ	0.99
ε	0.001

To assess the effectiveness of the P-DDPG algorithm, this paper compares its performance using the following techniques:

1. Local calculation (PCL): Each user doing the job performs it on the local CPU without offloading to the edge.
2. Random cache and computation offload (RCAO): The ratio between the caching and the offloading of tasks to the computer is randomized for each time slot of the MEC server until the capacity of the cache is reached.
3. DDQN: The selection and evaluation of actions are achieved through the use of different value functions, and tasks are cached and offloaded in an optimal ratio to achieve the lowest possible latency and power consumption.

5.2. Experimental Results and Analysis

5.2.1. Convergence Performance

Figure 3 compares the convergence performance of the P-DDPG and DDQN algorithms with respect to the average reward. They both use similar neural networks and can improve the performance of the system in terms of the MEC task caching and offloading. In the DDQN algorithm, discrete actions are mapped to continuous actions for performance, and an ε-greedy policy is used for action selection. In the experimental results, both algorithms converge, but there are large fluctuations in the initial phase because the reinforcement learning process is random, and the intelligent agent is in the exploration phase with low rewards. Increasing the number of training epochs, the intelligent agent enters the experience learning phase and has more time to explore the best strategy. As a result, the algorithms converge faster, and the reward values become stable. The P-DDPG and DDQN algorithms converge after about 250 and 360 training epochs, respectively. The P-DDPG algorithm converges to values of around -110, while the DDQN algorithm

converges to values of around −220. The system cost is inversely proportional to the reward. The P-DDPG algorithm consistently outperforms the DDQN algorithm in terms of the policies learned in different scenarios, indicating that the discarded action space affects performance. For the continuous control problem, the strategies of P-DDPG explore the action space more efficiently than those of DDQN. This demonstrates that the P-DDPG algorithm has faster convergence and better performance.

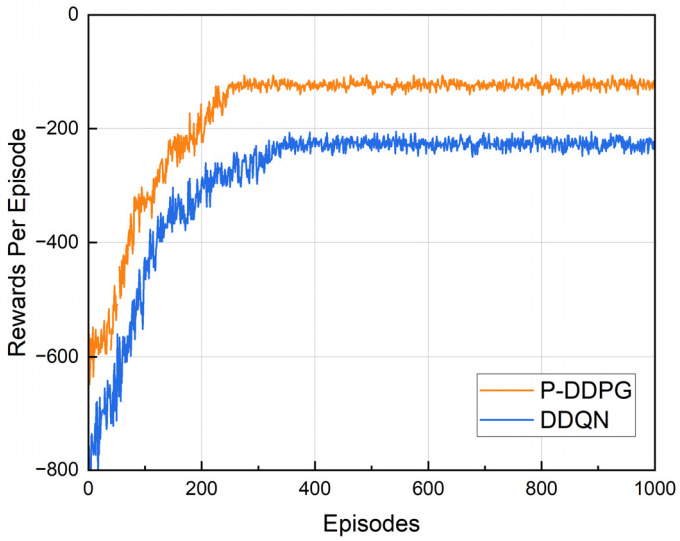

Figure 3. Comparison of convergence between the P-DDPG algorithm and DDQN algorithm.

5.2.2. Performance Comparison

Figures 4 and 5 show how the algorithms perform for different numbers of users. As the number of users grows from 4 to 12, the system latency and energy consumption of each algorithm increase. However, the system latency of the P-DDPG algorithm grows slower and is lower than the latency of the other algorithms. This indicates that, for a small number of users, different algorithms do not significantly impact the average task completion latency. As the number of users increases, offloading and computing more tasks result in larger system latency. Meanwhile, the server needs to reduce the computational resources allocated to each user, which leads to an increase in energy costs. The P-DDPG algorithm considers each task's computation to determine a caching and offloading policy that maintains low energy consumption even with few users. By implementing distributed decision-making across users, the P-DDPG algorithm significantly reduces the average system overhead and outperforms other benchmark algorithms.

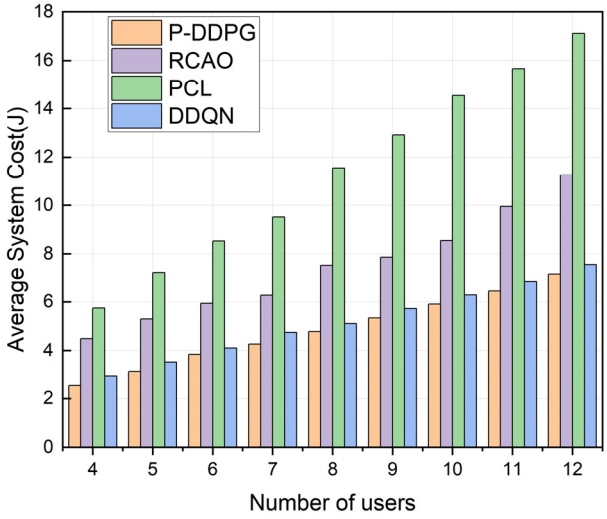

Figure 4. Average system cost for different numbers of users.

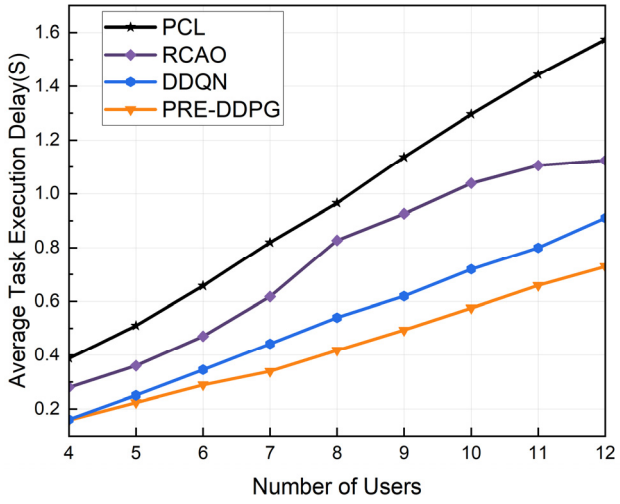

Figure 5. The average latency for different numbers of users.

Figure 6 demonstrates the impact of the computing power of the MEC server on the performance of the system. The system cost of the PCL remains constant because no MEC resources are used. The RCAO algorithm's strategy of randomly selecting each task leads to relatively poor performance due to its low computational complexity and strong randomness. Both the P-DDPG algorithm and the DDQN algorithm improve system performance, but as the MEC computational power increases, the performance advantage of the P-DDPG algorithm becomes more evident. The P-DDPG algorithm makes fuller use of the MEC resources and adaptively optimizes the offloading patterns, whereas the DDQN algorithm uses a very large action space because of the need to reduce the system cost, which makes the algorithm more complex. As a result, the P-DDPG algorithm consistently consumes less energy than the other three algorithms. Figure 7 shows the effect of average task data size on system cost. When the average task data size increases from 10 MB to 50 MB, more delays need to be handled, which leads to an increase in system cost. The DDQN algorithm requires more latency and energy to handle continuous actions. The P-DDPG algorithm can offload the tasks to the right location based on the characteristics of the task by learning the most valuable lessons. This enhances the efficiency of the edge nodes, which in turn improves the efficiency of caching and offloading tasks at the edge.

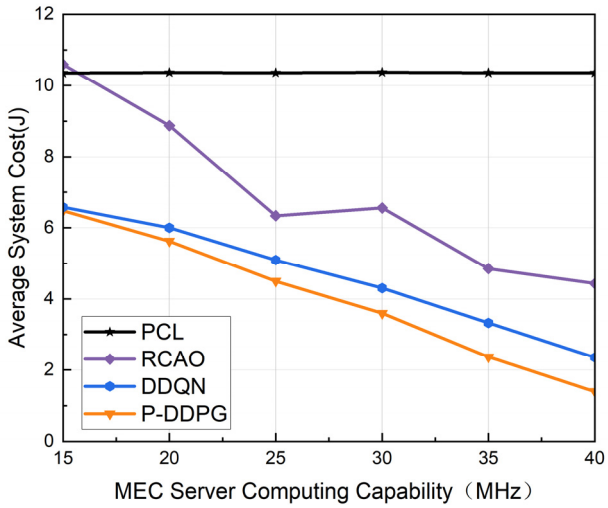

Figure 6. Average system cost with varying processing power.

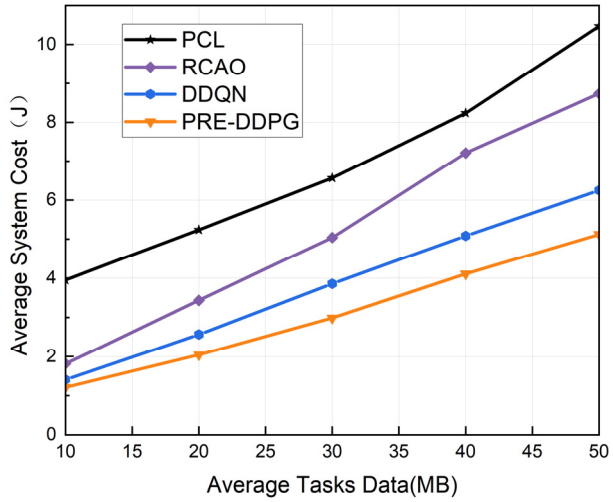

Figure 7. Average system cost for varying average task data sizes.

Figure 8 represents the effect of cache capacity on the performance of various caching schemes to further validate the effectiveness of the P-DDPG algorithm. The results show that as the cache capacity increases, the system cost decreases because a larger cache capacity can store more data and files so that the requested task has a higher hit rate on the MEC, which, in turn, reduces the cost of computing the task for the user. The RCAO algorithm adopts a random caching decision, which causes a larger proportion of popular files to be cached, leading to a higher retrieval cost compared to the DDQN algorithm and, thus, an increase in the system cost. The scheme proposed by the P-DDPG algorithm makes full use of the user's cache capacity, improves the cache utilization, and selects the operation with the highest return to reduce the computation by adaptively optimizing the offloading pattern, thereby reducing the system cost. Thus, better performance can be achieved by efficiently utilizing cache resources.

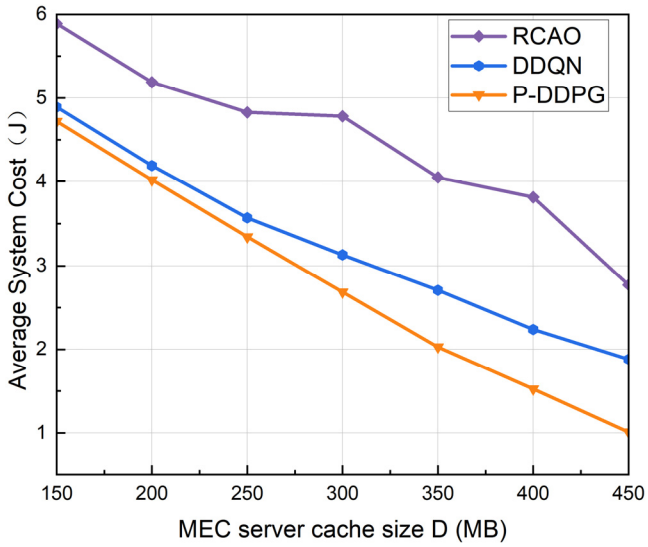

Figure 8. Average system cost for a range of cache sizes.

6. Conclusions

This paper addresses the joint problem of optimizing task caching and offloading for multiuser multitasking in MEC systems. Considering the storage and computation resource constraints of dynamic edge environments in the system, the user's mobility, and the change in resource states during multitasking offloading, a joint computation and caching framework was developed. This framework utilizes caching to assist in the reduction of redundant computations and transmissions between the mobile device and the MEC server. Then, the P-DDPG algorithm is designed to seek the optimal caching and offloading modes, improve the training stability of the algorithm, accelerate the training speed, and quickly obtain the optimal offloading strategy. This is achieved by introducing the PER and importance sampling mechanisms instead of the traditional experience replay caching and stochastic gradient descent methods. In future work, we plan to expand to multiple MEC servers, increasing the collaboration between task caches and balancing network overheads while improving the hit rate of tasks.

Author Contributions: Conceptualization, Z.J., X.Z., X.P. and S.Z.; methodology, Z.J., X.Z., X.P. and S.Z.; software, X.Z. and Z.J.; validation, X.Z., X.P. and S.Z.; formal analysis, X.Z.; investigation, Z.J. and X.Z.; resources, Z.J. and X.Z.; writing—original draft preparation, Z.J.; writing—review and editing, Z.J. and X.Z.; visualization, Z.J.; supervision, Z.J. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China Youth Fund (62202145).

Data Availability Statement: Dataset available on request from the authors.

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

References

1. Choi, Y.; Lim, Y. Edge Caching Based on Deep Reinforcement Learning in Vehicular Networks. In Proceedings of the 2022 IEEE 4th Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan, 28–30 October 2022; pp. 57–59.
2. Kiani, A.; Ansari, N. Edge Computing Aware NOMA for 5G Networks. *IEEE Internet Things J.* **2018**, *5*, 1299–1306. [CrossRef]
3. Tran, T.X.; Pompili, D. Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 856–868. [CrossRef]
4. Bi, S.; Huang, L.; Zhang, Y.J.A. Joint Optimization of Service Caching Placement and Computation Offloading in Mobile Edge Computing Systems. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 4947–4963. [CrossRef]
5. Wu, Z.; Yan, D. Deep reinforcement learning-based computation offloading for 5G vehicle-aware multi-access edge computing network. *China Commun.* **2021**, *18*, 26–41. [CrossRef]

6. Khurshid, T.; Ahmed, W.; Rehan, M.; Ahmad, R.; Alam, M.M.; Radwan, A. A DRL Strategy for Optimal Resource Allocation Along With 3D Trajectory Dynamics in UAV-MEC Network. *IEEE Access* **2023**, *11*, 54664–54678. [CrossRef]
7. Kong, X.; Duan, G.; Hou, M.; Shen, G.; Wang, H.; Yan, X.; Collotta, M. Deep Reinforcement Learning-Based Energy-Efficient Edge Computing for Internet of Vehicles. *IEEE Trans. Ind. Inform.* **2022**, *18*, 6308–6316. [CrossRef]
8. Xu, Z.; Zhou, L.; Dai, H.; Liang, W.; Zhou, W.; Zhou, P.; Xu, W.; Wu, G. Energy-Aware Collaborative Service Caching in a 5G-Enabled MEC With Uncertain Payoffs. *IEEE Trans. Commun.* **2022**, *70*, 1058–1071. [CrossRef]
9. Ren, J.; Yu, G.; He, Y.; Li, G.Y. Collaborative Cloud and Edge Computing for Latency Minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
10. Dai, H.; Wen, H.; Xing, H.; Ding, Z. Joint Cache Placement and NOMA-Based Task Offloading for Multi-User Mobile Edge Computing. In Proceedings of the 2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring), Florence, Italy, 20–23 June 2023; pp. 1–7.
11. Chen, Z.; Zhou, Z.; Chen, C. Code Caching-Assisted Computation Offloading and Resource Allocation for Multi-User Mobile Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4517–4530. [CrossRef]
12. Feng, H.; Guo, S.; Yang, L.; Yang, Y. Collaborative Data Caching and Computation Offloading for Multi-Service Mobile Edge Computing. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9408–9422. [CrossRef]
13. Tran, T.X.; Pompili, D. Adaptive Bitrate Video Caching and Processing in Mobile-Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2019**, *18*, 1965–1978. [CrossRef]
14. Shi, W.; Zhou, S.; Niu, Z.; Jiang, M.; Geng, L. Multiuser Co-Inference With Batch Processing Capable Edge Server. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 286–300. [CrossRef]
15. Chen, Z.; Yi, W.; Alam, A.S.; Nallanathan, A. Dynamic Task Software Caching-Assisted Computation Offloading for Multi-Access Edge Computing. *IEEE Trans. Commun.* **2022**, *70*, 6950–6965. [CrossRef]
16. Cheng, M.; Li, J.; Nazarian, S. DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, Republic of Korea, 22–25 January 2018.
17. Hazarika, B.; Singh, K.; Biswas, S.; Li, C.P. DRL-Based Resource Allocation for Computation Offloading in IoV Networks. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8027–8038. [CrossRef]
18. Li, S.; Li, B.; Zhao, W. Joint Optimization of Caching and Computation in Multi-Server NOMA-MEC System via Reinforcement Learning. *IEEE Access* **2020**, *8*, 112762–112771. [CrossRef]
19. Zhu, A.; Guo, S.; Ma, M.; Feng, H.; Liu, B.; Su, X.; Guo, M.; Jiang, Q. Computation Offloading for Workflow in Mobile Edge Computing Based on Deep Q-Learning. In Proceedings of the 2019 28th Wireless and Optical Communications Conference (WOCC), Beijing, China, 9–10 May 2019; pp. 1–5.
20. Liu, T.; Zhang, Y.; Zhu, Y.; Tong, W.; Yang, Y. Online Computation Offloading and Resource Scheduling in Mobile-Edge Computing. *IEEE Internet Things J.* **2021**, *8*, 6649–6664. [CrossRef]
21. Wang, L.; Zhang, G. Joint Service Caching, Resource Allocation and Computation Offloading in Three-Tier Cooperative Mobile Edge Computing System. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 3343–3353. [CrossRef]
22. Tang, H.; Wu, H.; Qu, G.; Li, R. Double Deep Q-Network Based Dynamic Framing Offloading in Vehicular Edge Computing. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 1297–1310. [CrossRef]
23. Zhou, H.; Zhang, Z.; Wu, Y.; Dong, M.; Leung, V.C.M. Energy Efficient Joint Computation Offloading and Service Caching for Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Green Commun. Netw.* **2023**, *7*, 950–961. [CrossRef]
24. Wang, Z.; Wu, T.; Zhang, Z.; Zhou, H. A Game theory-based Computation Offloading Method in Cloud-Edge Computing Networks. In Proceedings of the 2021 International Conference on Computer Communications and Networks (ICCCN), Athens, Greece, 19–22 July 2021; pp. 1–6.
25. Wu, Y.; Wang, Y.; Zhou, F.; Hu, R.Q. Computation Efficiency Maximization in OFDMA-Based Mobile Edge Computing Networks. *IEEE Commun. Lett.* **2020**, *24*, 159–163. [CrossRef]
26. Fan, W.; Han, J.; Su, Y.; Liu, X.; Wu, F.; Tang, B.; Liu, Y. Joint Task Offloading and Service Caching for Multi-Access Edge Computing in WiFi-Cellular Heterogeneous Networks. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 9653–9667. [CrossRef]
27. Xue, J.; An, Y. Joint Task Offloading and Resource Allocation for Multi-Task Multi-Server NOMA-MEC Networks. *IEEE Access* **2021**, *9*, 16152–16163. [CrossRef]
28. Mach, P.; Becvar, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
29. Nath, S.; Wu, J. Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems. *Intell. Converg. Netw.* **2020**, *1*, 181–198. [CrossRef]
30. Qiu, C.; Hu, Y.; Chen, Y.; Zeng, B. Deep Deterministic Policy Gradient (DDPG)-Based Energy Harvesting Wireless Communications. *IEEE Internet Things J.* **2019**, *6*, 8577–8588. [CrossRef]
31. Zhang, Z.; Zhouand, H.; Zhao, L.; Leung, V.C.M. Digital Twin Assisted Computation Offloading and Service Caching in Mobile Edge Computing. In Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), Bologna, Italy, 10–13 July 2022; pp. 1296–1297.

32. Zhou, H.; Wang, Z.; Zheng, H.; He, S.; Dong, M. Cost Minimization-Oriented Computation Offloading and Service Caching in Mobile Cloud-Edge Computing: An A3C-Based Approach. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 1326–1338. [CrossRef]
33. Wen, W.; Cui, Y.; Quek, T.Q.S.; Zheng, F.C.; Jin, S. Joint Optimal Software Caching, Computation Offloading and Communications Resource Allocation for Mobile Edge Computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 7879–7894. [CrossRef]
34. Niu, J.; Zhang, S.; Chi, K.; Shen, G.; Gao, W. Deep learning for online computation offloading and resource allocation in NOMA. *Comput. Netw.* **2022**, *216*, 109238. [CrossRef]
35. He, X.; Lu, H.; Du, M.; Mao, Y.; Wang, K. QoE-Based Task Offloading With Deep Reinforcement Learning in Edge-Enabled Internet of Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 2252–2261. [CrossRef]
36. Xue, Z.; Liu, C.; Liao, C.; Han, G.; Sheng, Z. Joint Service Caching and Computation Offloading Scheme Based on Deep Reinforcement Learning in Vehicular Edge Computing Systems. *IEEE Trans. Veh. Technol.* **2023**, *72*, 6709–6722. [CrossRef]