sensors

MDPI

Article

Computing Offloading Based on TD3 Algorithm in Cache-Assisted Vehicular NOMA–MEC Networks

Tianqing Zhou [1], Ming Xu [1], Dong Qin [2,*], Xuefang Nie [1], Xuan Li [1] and Chunguo Li [3]

[1] School of Information Engineering, East China Jiaotong University, Nanchang 330013, China; zhoutian930@163.com (T.Z.); xm1020487915@163.com (M.X.); xuefangnie@163.com (X.N.); lixuan@ecjtu.edu.cn (X.L.)
[2] School of Information Engineering, Nanchang University, Nanchang 330031, China
[3] School of Information Science and Engineering, Southeast University, Nanjing 210096, China; chunguoli@seu.edu.cn
* Correspondence: qindong@ncu.edu.cn; Tel.: +86-157-9789-6518

Abstract: In this paper, in order to reduce the energy consumption and time of data transmission, the non-orthogonal multiple access (NOMA) and mobile edge caching technologies are jointly considered in mobile edge computing (MEC) networks. As for the cache-assisted vehicular NOMA–MEC networks, a problem of minimizing the energy consumed by vehicles (mobile devices, MDs) is formulated under time and resource constraints, which jointly optimize the computing resource allocation, subchannel selection, device association, offloading and caching decisions. To solve the formulated problem, we develop an effective joint computation offloading and task-caching algorithm based on the twin-delayed deep deterministic policy gradient (TD3) algorithm. Such a TD3-based offloading (TD3O) algorithm includes a designed action transformation (AT) algorithm used for transforming continuous action space into a discrete one. In addition, to solve the formulated problem in a non-iterative manner, an effective heuristic algorithm (HA) is also designed. As for the designed algorithms, we provide some detailed analyses of computation complexity and convergence, and give some meaningful insights through simulation. Simulation results show that the TD3O algorithm could achieve lower local energy consumption than several benchmark algorithms, and HA could achieve lower consumption than the completely offloading algorithm and local execution algorithm.

Keywords: TD3; MEC; NOMA; vehicular networks; edge cache; computation offloading; resource allocation

Citation: Zhou, T.; Xu, M.; Qin, D.; Nie, X.; Li, X.; Li, C. Computing Offloading Based on TD3 Algorithm in Cache-Assisted Vehicular NOMA–MEC Networks. *Sensors* **2023**, *23*, 9064. https://doi.org/10.3390/s23229064

Academic Editors: Pingyi Fan and Qiong Wu

Received: 10 October 2023
Revised: 2 November 2023
Accepted: 7 November 2023
Published: 9 November 2023

1. Introduction

With the rapid development of information and communication technologies, the data traffic generated by vehicles (mobile devices, MDs) has also significantly increased [1]. For wireless communication networks, more spectrum resources are required for data traffic transmission [2–5]. In addition, higher computing power is required by MDs for supporting large amounts of task calculation. However, due to the limited battery capacity of MDs, it may be challenging to process these computation tasks for them. By deploying edge computing servers at base stations (BSs), mobile edge computing (MEC) can support MDs in processing tasks at the adjacent edge servers [6,7]. Compared with cloud computing (CC), which requires tasks to be uploaded to a remote cloud, MEC can provide additional computing resources for MDs within its coverage area and thus reduce their computing overhead [8–14].

Although the edge servers can reduce the computing overhead of MDs by providing more computing resources, the extra time and energy consumption caused by offloading tasks through wireless channels cannot be ignored, especially for high-size computation tasks. In order to further reduce the time and energy consumption caused by offloading tasks, edge caching technology is also introduced into MEC networks. By caching tasks of

MDs at edge servers in advance, the overhead caused by offloading tasks could be greatly reduced [15–19].

To upload tasks from MDs to edge servers, orthogonal multiple access (OMA) is often used, but it may be greatly challenging to provide a high transmission rate and support massive connections. As another type of resource utilization management, non-orthogonal multiple access (NOMA) technologies can let multiple users share the same frequency bands, achieve higher spectral efficiency and support extensive connections [20–23]. It is evident that NOMA is a good type of resource utilization management for reducing the cost of task transmission in MEC networks.

Although the application of caching and NOMA technologies in MEC networks can reduce time and energy consumption, such a framework will make the design of computation offloading and edge caching schemes more complex. To the best of our knowledge, until now, how to jointly perform the device association, computation offloading, edge caching, subchannel selection and resource allocation is still an important and open topic in cache-assisted NOMA–MEC networks.

1.1. Related Work

So far, a lot of work has been conducted on joint computation offloading and resource optimization in NOMA–MEC networks. In [20], joint radio and computation resource allocation was optimized to maximize the offloading energy efficiency in NOMA–MEC-enabled IoT networks, and a solution based on a multi-layer iterative algorithm was proposed. In [21], local computation resource, offloading ratio, uplink transmission time and power and subcarrier assignment were jointly optimized to minimize the sum of weighted energy consumed by users in NOMA–MEC networks, and some effective iterative algorithms were designed for single-user and multi-user cases. In [24], joint task offloading, power allocation and computing resource allocation were optimized to achieve delay minimization using a deep reinforcement learning (DRL) algorithm in NOMA–MEC networks. In [25], the joint optimization of offloading decisions, local and edge computing resource allocation and power and subchannel allocation were realized to minimize energy consumption in heterogeneous NOMA–MEC networks, and an effective iterative algorithm was designed. In [26], power and computation resource allocations were jointly optimized to minimize overall computation and transmission delay for massive MIMO and NOMA-assisted MEC systems, and a solution based on an interior-point algorithm was given. In [27], the channel resource allocation and computation offloading policy were jointly optimized to minimize the sum of weighted energy and latency in NOMA–MEC networks, and some efficient solutions were found using a DRL algorithm based on actor–critic and deep Q-network (DQN) methods.

To further reduce the offloading time and energy consumption, edge caching technology is introduced into conventional MEC networks. Such a framework has attracted more and more attention. In [28], the offloading and caching decisions, uplink power and edge computing resources were jointly optimized to minimize the sum of weighted local processing time and energy consumption in two-tier cache-assisted MEC networks, and a distributed collaborative iterative algorithm was proposed. In [29], a problem of adaptive request scheduling and cooperative service caching was studied in cache-assisted MEC networks. After formulating the optimization problems as partially observable Markov decision process (MDP) problems, an online DRL algorithm was proposed to improve the service hitting ratio and latency reduction rate. In [30], optimal offloading and caching strategies were established to minimize overall delay and energy consumption of all regions using a deep deterministic policy gradient (DDPG) framework in cache-assisted multi-region MEC networks. In [31], joint MD association and resource allocation were performed to minimize the sum of MDs' weighted delay in heterogeneous cellular networks with MEC and edge caching functions, and an effective iterative algorithm was developed using coalitional game and convex optimization theorems. In [32], to minimize the content transmission delay in vehicular edge computing networks, a cooperative vehicular edge

computing and caching scheme based on asynchronous federated and deep reinforcement learning was proposed to predict the popular content and the optimal cooperative caching location of the content. In [33], to reduce the cost of the cloud service center through the asynchronous advantage actor–critic algorithm, the offloading decision, service caching and resource allocation strategies were jointly optimized in the three-tier mobile cloud–edge computing structure combining computation offloading and service caching mechanisms. In [34], a logistic function-based service reliability probability (SRP) estimation model was built, and the average SRP maximization problem of a virtual machine-based edge computing server was studied for such a model. At last, a low-complexity heuristic alternative optimization algorithm was proposed.

To enhance spectral efficiency and support massive connections, NOMA technology has attracted increasing attention in cache-assisted MEC networks. In [35], the multi-agent deep deterministic policy gradient method was used to dynamically optimize the user association, power control and cache placement of BSs and satellites to improve the network energy efficiency in a NOMA-enabled satellite integrated with a terrestrial network scenario. In [36], joint optimization of offloading and caching decisions and computation resource allocation was performed to maximize long-term reward in cache-assisted NOMA–MEC networks under the predicted task popularity, and single-agent and multi-agent Q-learning algorithms were proposed to find feasible solutions. In [37], joint optimization of offloading and caching decisions was performed to minimize the system delay in cache-assisted NOMA–MEC networks, and a multi-agent DQN algorithm was used for finding efficient solutions under the predicted popularity. In [38], local task processing time was minimized by jointly optimizing offloading and caching decisions and the allocation of edge computing resources and uplink power in cache-assisted NOMA–MEC networks with single BS, and the blocking successive upper-bound minimization method was utilized to achieve efficient solutions.

Although the framework of cache-assisted (vehicular) NOMA–MEC networks can greatly reduce the task processing time and energy consumption and support massive connections, there exist very few relevant efforts. Unlike the above-mentioned work, we jointly optimize the edge computing resource allocation, subchannel selection, device association, offloading and caching decisions for the cache-assisted vehicular NOMA–MEC networks with multiple BSs, minimizing the energy consumed by MDs under time and resource constraints. In addition, unlike existing efforts, we develop an effective dynamic joint computation offloading and task-caching algorithm based on the twin-delayed deep deterministic policy gradient algorithm (TD3) to find efficient solutions, named the TD3-based offloading (TD3O) algorithm.

1.2. Contribution and Organization

In this paper, we jointly optimize the edge computing resource allocation, subchannel selection, device association, offloading and caching decisions in cache-assisted vehicular NOMA–MEC networks, minimizing the energy consumed by MDs under time and resource constraints. Specifically, the main contributions and work of this paper can be listed as follows.

- Edge computing resource allocation, subchannel selection, device association, computation offloading and edge caching are jointly performed in cache-assisted vehicular NOMA–MEC networks. To the best of our knowledge, work that concerns subchannel selection is a new investigation for cache-assisted vehicular NOMA–MEC networks with multi-server scenarios. Meanwhile, as far as this problem is concerned, the goal is to minimize the energy consumed by MDs under the constraints of time, computing resources, caching capacity, the number of MDs associated with each BS and the number of MDs associated with each subchannel. As far as we know, such an optimization problem is a new concentration in cache-assisted vehicular NOMA–MEC networks.
- We design effective algorithms to find feasible solutions to the formulated problem. Considering that the formulated problem is in a mixed-integer, nonlinear, multi-

constraint form, a simple map between actions and actual policies in a conventional twin-delayed deep deterministic policy gradient (TD3) algorithm cannot be well applied. In addition, too large an action space will cause the TD3 algorithm to fail to search for correct actions and thus fail to converge. In view of these concerns, we develop an effective TD3O algorithm integrating with the AT algorithm to solve the formulated problem. Moreover, in order to solve this problem in a non-iterative manner, an effective heuristic algorithm (HA) is also designed.

- Performance analyses of the designed algorithms. Some analyses are made for the computation complexity and convergence of the designed algorithms in detail. In addition, some meaningful simulation analyses are also made by introducing other benchmark algorithms for comparison, and some good results and insights are achieved.

The rest of the paper is organized as follows. Section 2 introduces the system model. Section 3 formulates a problem of minimizing local energy consumption in cache-assisted vehicular NOMA–MEC networks. Section 4 designs the HA and TD3O algorithm. Section 5 gives the computation complexity and convergence analyses for the designed algorithms. Section 6 investigates the performance of the designed algorithms through simulation. Section 7 gives conclusions and discussions.

2. System Model

2.1. Network Model

Figure 1 shows the cache-assisted vehicular NOMA–MEC networks. In such network, there exist M MDs, and the index set of them is denoted as $\mathcal{M} = \{1, 2, \cdots, M\}$; B BSs are deployed, and the index set of them is given by $\mathcal{I} = \{1, 2, \cdots, I\}$. In addition, each BS is equipped with one edge computing server and one edge caching server, and these BSs connect to each other through wired links. We assume that each MD has one computation task at any timeslot, which can be processed by itself, its associated BS or another auxiliary BS selected by this associated BS. When tasks have been cached at the BSs used for processing them, they do not need to be uploaded to these BSs; when the associated BSs have not cached tasks, MDs need to upload tasks to these BSs; when the auxiliary BSs have not cached tasks, the associated BSs need to upload tasks to their selected auxiliary BSs.

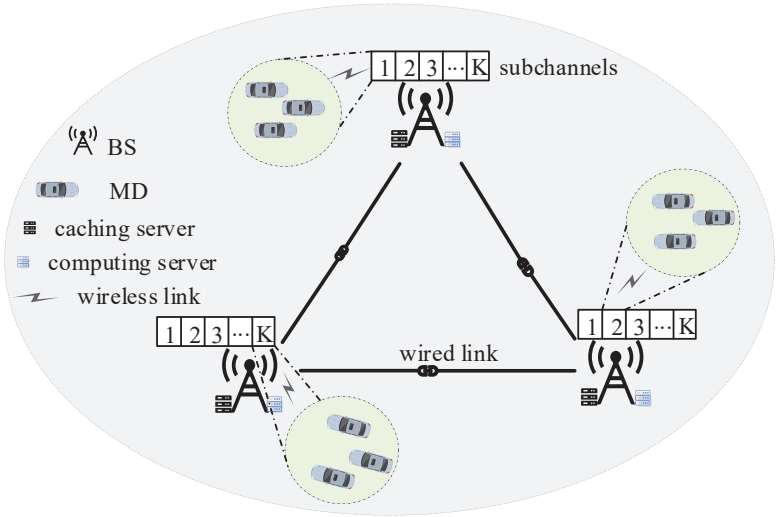

Figure 1. Cache-assisted vehicular NOMA–MEC networks.

Assume that the association index between MD m and BS i is $x_{m,i} \in \{0, 1\}$, where $\mathbf{X} = \{x_{m,i} | \forall m \in \mathcal{M}, \forall i \in \mathcal{I}\}$. $x_{m,i} = 1$ if MD m is associated with BS i, otherwise $x_{m,i} = 0$. In addition, we assume that the caching index of the task of MD m at BS i is denoted as $y_{m,i} \in \{0, 1\}$, where $\mathbf{Y} = \{y_{m,i} | \forall m \in \mathcal{M}, \forall i \in \mathcal{I}\}$. $y_{m,i} = 1$ if the task of MD m is cached at BS i, otherwise $y_{m,i} = 0$. We also assume that the offloading (execution) in-

dex of the task of MD m at BS i is denoted as $u_{m,i}$, where $\mathbf{U} = \{u_{m,i} | \forall m \in \mathcal{M}, \forall i \in \mathcal{I}\}$. $u_{m,i} = 1$ if the task of MD m is executed at BS i, otherwise $u_{m,i} = 0$. At last, we assume that the association index between MD m and subchannel k of BS i is denoted as $z_{m,i,k}$, where $\mathbf{Z} = \{z_{m,i,k} | \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall k \in \mathcal{K}\}$. If $x_{m,i}(1 - y_{m,i})(1 - y_{m,\bar{i}})u_{m,\bar{i}} = 1$ or $x_{m,i}(1 - y_{m,i})u_{m,i} = 1$ under $\bar{i} \neq i$, MD m can select (be associated with) some subchannel k of BS i, which means $z_{m,i,k} = 1$. Otherwise, the subchannel k of BS i cannot be selected by MD m, which means $z_{m,i,k} = 0$.

2.2. Communication Model

In this paper, the system bandwidth W is divided into K subchannels with equal bandwidth, which are indexed by $\mathcal{K} = \{1, 2, \cdots, K\}$. These subchannels can be shared by different MDs through NOMA. Significantly, each MD can occupy at most one subchannel, the number of MDs selecting each subchannel cannot exceed the upper limit ρ, and the number of MDs associated with any BS that need to upload tasks should be less than or equal to the number of subchannels K [37].

As revealed in [39], the successive interference cancellation (SIC) technology in NOMA technology can effectively reduce the interference between MDs in the same subchannel. The channel gains of MDs sharing the same subchannel of a BS should be sorted in descending order at first, and then the uplink NOMA signals received by this BS can be decoded in this order. We assume that $\mathcal{M}_k^{SC}$ is the set of MDs selecting subchannel k, and $o_{m,i,k}$ represents the sequence number of channel gain between MD m and BS i on subchannel k. When MD i and MD m access the subchannel k of BS i simultaneously and the channel gain $h_{j,i,k}$ between MD j and BS i on subchannel k is lower than the channel gain $h_{m,i,k}$ between MD m and BS i on subchannel k, $o_{j,i,k} < o_{m,i,k}$ is satisfied. Then, the signal of MD m is decoded but the signal of MD j will be treated as noise. Therefore, when MD m selects subchannel k of BS i, its uplink data rate $r_{m,i,k}$ can be given by

$$r_{m,i,k} = W \log_2 \left(1 + p_m h_{m,i,k} / \left(\Gamma_{m,i,k} + \sigma^2\right)\right) / K, \tag{1}$$

where $\Gamma_{m,i,k} = \sum_{j \in \mathcal{M}_k^{sc} / \{m\} : o_{j,i,k} < o_{m,i,k}} p_j h_{j,i,k}$ is the interference caused by other MDs (excluding MD m) sharing subchannel k of BS i through NOMA; p_m is the transmission power of MD m; σ^2 is the noise power. When MD m is decoded, it is no longer regarded as interference in the subchannel, and the device with the maximum channel gain among the remaining MDs in the current subchannel is decoded in the same way until all MDs of the current subchannel are decoded.

2.3. Caching and Offloading Models

In this paper, we assume that any MD m has a time-sensitive task denoted as $\mathcal{L}_m = \{d_m, c_m, \tau_m^{\max}\}$ at each timeslot, where d_m is the data size of the task of MD m, c_m is the number of CPU cycles required to complete a one-bit task, and $\tau_m^{\max}$ is the maximum task processing time of MD m.

Figure 2 illustrates the caching and offloading models. At each timeslot, BSs precache the tasks for processing at the next timeslot. When MD m is associated with BS i, it first checks whether the associated BS has cached the corresponding task. If $\sum_{i \in \mathcal{I}} u_{m,i} = 0$, the task of MD m is calculated by itself, e.g., MD 1 in Figure 2. If $x_{m,i} y_{m,i} u_{m,i} = 1$, the task of MD m can be directly calculated at its associated BS i, and the results will be fed back from BS i to MD m, e.g., MD 2 in Figure 2. If $x_{m,i} y_{m,i} (1 - y_{m,\bar{i}}) u_{m,\bar{i}} = 1$, the task of MD m is offloaded from its associated BS i to another auxiliary BS $\bar{i} \neq i$ for computing through a wired link, e.g., MD 3 in Figure 2. If $x_{m,i} y_{m,i} y_{m,\bar{i}} u_{m,\bar{i}} = 1$, the task of MD m can be directly calculated at auxiliary BS $\bar{i} \neq i$, e.g., MD 4 in Figure 2. If $x_{m,i}(1 - y_{m,i}) u_{m,i} = 1$, the task of MD m will be offloaded to its associated BS i for computing, e.g., MD 5 in Figure 2. If $x_{m,i}(1 - y_{m,i})(1 - y_{m,\bar{i}}) u_{m,\bar{i}} = 1$, the task of MD m first needs to be offloaded to its associated BS i, and then it is transmitted from this BS to another auxiliary BS $\bar{i} \neq i$

for computing through a wired link, e.g., MD 6 in Figure 2. If $x_{m,i}(1-y_{m,i})y_{m,\bar{i}}u_{m,\bar{i}}=1$, the task of MD m can be directly calculated at an auxiliary BS $\bar{i}\neq i$, e.g., MD 4 in Figure 2.

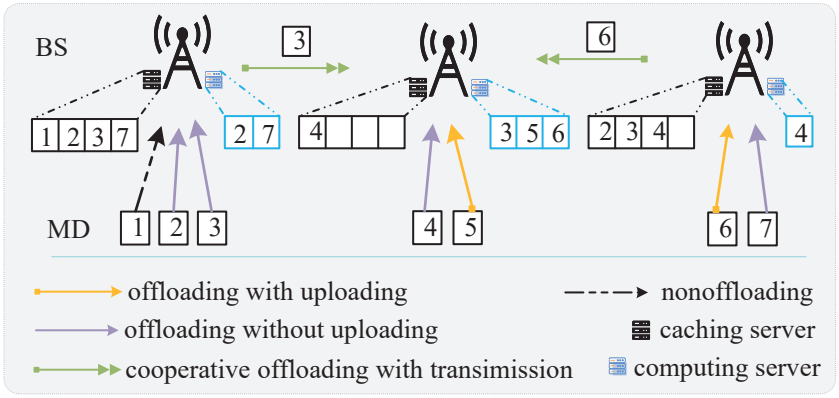

Figure 2. Caching and offloading models.

2.3.1. Local Computing

If $\sum_{i\in\mathcal{I}}u_{m,i}=0$ is satisfied, the task of MD m should be executed locally, and the processing time and energy consumption are, respectively, given by

$$\tau_m^{\mathrm{loc}}=c_m d_m\Big/f_m^{\mathrm{loc}}, \tag{2}$$

$$\varepsilon_m^{\mathrm{loc}}=\xi c_m d_m\left(f_m^{\mathrm{loc}}\right)^2, \tag{3}$$

where f_m^{loc} is the computing capacity of MD m, and ξ is an energy-consumption coefficient depending on the hardware architecture.

2.3.2. Task Transmission

If $x_{m,i}(1-y_{m,i})(1-y_{m,\bar{i}})u_{m,\bar{i}}=1$ or $x_{m,i}(1-y_{m,i})u_{m,i}=1$ are satisfied under $\bar{i}\neq i$, the task of MD m should be, respectively, uploaded to BS $\bar{i}$ or i for execution through NOMA. Then, the uploading time and energy consumption of MD m, respectively, are given by

$$\tau_m^{\mathrm{trs}}=\sum\nolimits_{i\in\mathcal{I}}\sum\nolimits_{k\in\mathcal{K}}z_{m,i,k}d_m/r_{m,i,k}, \tag{4}$$

$$\varepsilon_m^{\mathrm{trs}}=p_m\tau_m^{\mathrm{trs}}. \tag{5}$$

In addition, if $x_{m,i}(1-y_{m,i})(1-y_{m,\bar{i}})u_{m,\bar{i}}=1$ or $x_{m,i}y_{m,i}(1-y_{m,\bar{i}})u_{m,\bar{i}}=1$ is satisfied under $\bar{i}\neq i$, the task of MD m should be transmitted from its associated BS i to an auxiliary BS $\bar{i}$ through a wired link, and the corresponding time is given by

$$\tau_m^{\mathrm{bh}}=d_m\Big/r^{\mathrm{bh}}, \tag{6}$$

where r^{bh} is the backhualing rate between any two BSs.

In this paper, we mainly concentrate on the energy consumption of MDs but not the energy consumed by BSs. In addition, the downlink transferring time of results is often ignored since they are fairly small [40].

2.3.3. Edge Computing

When MD m executes its task at BS i, the task processing time at this BS can be given by

$$\tau_{m,i}^{\mathrm{exe}}=c_m d_m/f_{m,i}, \tag{7}$$

where $f_{m,i}$ is the computing capacity allocated to MD m by BS i.

2.3.4. Task Processing Time and Energy Consumption

Then, the total time used for processing the task of MD m can be given by

$$\begin{aligned}\tau_m^{\text{tot}} = \sum_{i\in\mathcal{I}} \Big(& \big(1-\sum_{i\in\mathcal{I}} u_{m,i}\big)\tau_m^{\text{loc}} \\ & + x_{m,i}\big(1-y_{m,i}\big)\sum_{\bar{i}\in\mathcal{I}\setminus\{i\}} u_{m,\bar{i}}\big(1-y_{m,\bar{i}}\big)\tau_m^{\text{trs}} \\ & + x_{m,i}\big(1-y_{m,i}\big)\sum_{\bar{i}\in\mathcal{I}\setminus\{i\}} u_{m,\bar{i}}\big(1-y_{m,\bar{i}}\big)\tau_m^{\text{bh}} \\ & + x_{m,i}\big(1-y_{m,i}\big)\sum_{\bar{i}\in\mathcal{I}\setminus\{i\}} u_{m,\bar{i}}\tau_{m,\bar{i}}^{\text{exe}} \\ & + x_{m,i}y_{m,i}\sum_{\bar{i}\in\mathcal{I}\setminus\{i\}} u_{m,\bar{i}}\big(1-y_{m,\bar{i}}\big)\tau_m^{\text{bh}} \\ & + x_{m,i}y_{m,i}\sum_{\bar{i}\in\mathcal{I}\setminus\{i\}} u_{m,\bar{i}}\tau_{m,\bar{i}}^{\text{exe}} \\ & + x_{m,i}\big(1-y_{m,i}\big)u_{m,i}\big(\tau_m^{\text{trs}}+\tau_{m,i}^{\text{exe}}\big) \\ & + x_{m,i}y_{m,i}u_{m,i}\tau_{m,i}^{\text{exe}}\Big).\end{aligned} \tag{8}$$

On the right side of the equality sign in (8), the first item represents the local executing time; the second item is the time used for uploading the task from MD m to the associated BS i, which does not cache this task and further transmits it to auxiliary BS for computing; the third item is the time used for transmitting the task from the associated BS i to another auxiliary BS, where these two BSs do not cache this task; the fourth item is the time used for executing the task of MD m at an auxiliary BS, where the associated BS does not cache this task; the fifth item is the time used for transmitting the task from the associated BS i to another auxiliary BS, where the associated BS caches this task but the auxiliary BS does not; the sixth item is the time used for executing the task of MD m at an auxiliary BS, where the associated BS caches this task; the seventh item includes the time used for transmitting the task from MD m to the associated BS i, which does not cache this task, and the time used for executing the task of MD m at this BS; the eighth item is the time used for executing the task of MD m at the associated BS i, which caches this task.

Then, the total local energy consumption used for processing the task of MD m can be given by

$$\begin{aligned}\varepsilon_m^{\text{tot}} = \sum_{i\in\mathcal{I}} \Big(& \big(1-\sum_{i\in\mathcal{I}} u_{m,i}\big)\varepsilon_m^{\text{loc}} \\ & + x_{m,i}\big(1-y_{m,i}\big)\sum_{\bar{i}\in\mathcal{I}\setminus\{i\}} u_{m,\bar{i}}\big(1-y_{m,\bar{i}}\big)\varepsilon_m^{\text{trs}} \\ & + x_{m,i}\big(1-y_{m,i}\big)u_{m,i}\varepsilon_m^{\text{trs}}\Big),\end{aligned} \tag{9}$$

on the right side of equality sign in (9), the first item represents the local executing energy consumption; the second item is the energy consumption caused by offloading the task from MD m to its associated BS i, which further transmits this task to auxiliary BS $\bar{i} \neq i$ for computing; the third item is the energy consumption caused by transmitting the task from MD m to the associated BS i, which does not cache this task.

3. Problem Formulation

Until now, we have formulated a problem of minimizing local energy consumption at each given period. Specifically, under the constraints of time, computing resources, caching capacity, the number of MDs associated with each BS and the number of MDs associated with each subchannel, we jointly optimized the edge computing resource allocation, subchannel selection, device association, offloading and caching decisions to minimize the energy consumed by MDs in cache-assisted vehicular NOMA–MEC networks. Mathematically, this is formulated as

$$
\begin{aligned}
\mathrm{P1}: & \min_{\mathbf{X},\mathbf{Y},\mathbf{U},\mathbf{Z},\mathbf{F}} \sum_{m\in\mathcal{M}} \varepsilon_m^{\mathrm{tot}} \\
\text{s.t. } & C_1: \tau_m^{\mathrm{tot}} \le \tau_m, \forall m \in \mathcal{M}, \\
& C_2: \sum_{i\in\mathcal{I}} x_{m,i} = 1, \forall m \in \mathcal{M}, \\
& C_3: \sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{K}} z_{m,i,k} \le 1, \forall m \in \mathcal{M}, \\
& C_4: \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} z_{m,i,k} \le K, \forall i \in \mathcal{I}, \\
& C_5: \sum_{i\in\mathcal{I}} u_{m,i} \le 1, \forall m \in \mathcal{M}, \\
& C_6: x_{m,i} \in \{0,1\}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \\
& C_7: y_{m,i} \in \{0,1\}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \\
& C_8: z_{m,i,k} \in \{0,1\}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \forall k \in \mathcal{K}, \\
& C_9: u_{m,i} \in \{0,1\}, \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \\
& C_{10}: \sum_{m\in\mathcal{M}} y_{m,i} d_m \le \vartheta_i, \forall i \in \mathcal{I}, \\
& C_{11}: \sum_{m\in\mathcal{M}} \sum_{i\in\mathcal{I}} z_{m,i,k} \le \rho, \forall k \in \mathcal{K}, \\
& C_{12}: \sum_{m\in\mathcal{M}} u_{m,i} f_{m,i} \le f_i^{\mathrm{BS}}, \forall i \in \mathcal{I},
\end{aligned}
\tag{10}
$$

where $\mathbf{F} = \{f_{m,i} | \forall m \in \mathcal{M}, \forall i \in \mathcal{I}\}$; the constraint C_1 gives the maximum task processing time of MD m; C_2 and C_6 indicate that any MD m can select only one BS; C_3 and C_8 indicate that any MD m can occupy at most one subchannel; C_4 and C_8 mean that the number of MDs selecting any BS that needs to upload tasks should be less than or equal to the number of subchannels; C_5 and C_9 mean that any MD m can select at most one BS to execute its task; C_7 and C_{10} indicate that the data size of tasks cached at BS i does not exceed the caching capacity ϑ_i of this BS; C_8 and C_{11} show that the number of MDs selecting a subchannel cannot exceed its upper limit; C_7 and C_{12} reveal that the total computing capacity allocated to MDs by BS i cannot exceed the computing capacity of this BS.

4. Algorithm Design

As previously mentioned, the optimization problem P1 refers to minimizing local energy consumption within a given period. In view of this, we adopt the DRL algorithm to solve it. DRL is based on MDP, which implements the environment-based output of agent policy in MDP through neural networks, maximizing certain rewards. Considering that the overestimation of some conventional DRL algorithms (e.g., DQN and DDPG), the TD3 algorithm has been widely advocated because it can overcome well the problems of the above algorithms and achieve more stable output [41,42]. The main features of the TD3 algorithm are adding a new neural network and reducing the training frequency of the network based on DDPG.

The problem P1 has both continuous and discrete variables, and the solution space formed by the combination of all variables is very large, which is not a suitable scenario for the DQN algorithm to solve discrete space problems. Therefore, we use the TD3 algorithm, which can solve the continuous solution space problems. Considering that a simple mapping between only the decision of the algorithm and the actual strategy will fail to achieve convergence because of there being too many feasible strategies and the inability to search for the correct one, we develop an effective TD3O algorithm integrated with the AT algorithm to solve the problem P1.

4.1. MDP Used for Describing Problem P1

Considering that the optimization problem P1 needs to be tackled within a given period, in order to apply TD3O to the problem P1, such a period is divided into T timeslots and denoted as $\mathcal{T} = \{1, 2, \cdots, T\}$. Furthermore, the problem of joint computing offloading, task caching and resource allocation is described as a MDP, the state space, action space and reward function are defined as follows.

❶ State space: At each timeslot, the state space contains the information used for decisions made by the network. Here, the state s_t at timeslot t can be denoted as $s_t = \{\bar{\mathbf{D}}(t+1), \bar{\mathbf{Y}}(t)\}$. The detailed definitions can be found as follows.

- $\bar{\mathbf{D}}(t+1) = \{\bar{d}_m(t+1)|\forall m \in \mathcal{M}\}$ are the standardized data sizes of tasks of MDs at timeslot $t+1$, where
$$\bar{d}_m(t) = \frac{d_m(t) - d^{\min}(t)}{d^{\max}(t) - d^{\min}(t)}, \tag{11}$$
$d^{\min}(t)$ is the minimum data size of the tasks of all MDs at timeslot t, and $d^{\max}$ is the maximum data size of the tasks of all MDs at timeslot t.
- $\bar{\mathbf{Y}}(t) = \{\bar{y}_m(t)|\forall m \in \mathcal{M}\}$ are the task caching decision factors at BSs at timeslot t, where $\bar{y}_m \in [0,1]$.

❷ Action space: At each timeslot, the action space refers to the decisions made by the network according to the state s_t. The action a_t at timeslot t can be denoted as $a_t = \{\bar{\mathbf{X}}(t), \bar{\mathbf{Y}}(t+1), \bar{\mathbf{Z}}(t), \bar{\mathbf{U}}(t), \bar{\mathbf{F}}(t)\}$. Specifically:

- $\bar{\mathbf{X}}(t) = \{\bar{x}_m(t)|\forall m \in \mathcal{M}\}$ are the association decision factors of MDs at timeslot t, where $\bar{x}_m \in [0,1]$.
- $\bar{\mathbf{Y}}(t+1) = \{\bar{y}_m(t+1)|\forall m \in \mathcal{M}\}$ are the caching decision factors at timeslot t for the next timeslot.
- $\bar{\mathbf{Z}}(t) = \{\bar{z}_i(t)|\forall i \in \mathcal{I}\}$ are the subchannel allocation decision factors of BSs at timeslot t, where $\bar{z}_i \in [0,1]$.
- $\bar{\mathbf{U}}(t) = \{\bar{u}_m(t)|\forall m \in \mathcal{M}\}$ are the offloading decision factors of MDs at timeslot t, where $\bar{u}_m \in [0,1]$.
- $\bar{\mathbf{F}}(t) = \{\bar{f}_m(t)|\forall m \in \mathcal{M}\}$ are the computing resource allocation factors of MDs at timeslot t, where $\bar{f}_m \in [0,1]$.

It is noteworthy that the dimensions of the above-mentioned state and action spaces have been greatly reduced compared to the actual ones. The actual state and action spaces can be achieved by executing an AT algorithm in the following parts.

❸ Reward: Considering that the goal of problem P1 is to minimize local energy consumption and the constraints C_1 and C_{10} cannot be strictly satisfied in the DRL-based iteration procedure, the reward w_t at timeslot t is given by

$$w_t = -\omega_1 \sum\nolimits_{m\in\mathcal{M}} \varepsilon_m^{\text{tot}}(t) - \omega_2\phi(t) - \omega_3\varphi(t), \tag{12}$$

where $\phi(t) = \sum_{m\in\mathcal{M}} \max(\tau_m^{\text{tot}}(t) - \tau_m, 0)$ is the penalty function added for guaranteeing the constraint C_1; $\varphi(t) = \sum_{i\in\mathcal{I}} \max(\sum_{m\in\mathcal{M}} y_{m,i}(t) d_m(t) - \vartheta_i(t), 0)$ is the penalty function introduced for guaranteeing the constraint C_{10}; ω_1 is the energy-consumption discount factor; ω_2 and ω_3 are penalty coefficients.

When the network obtains action a_t according to the state s_t, the state space will obtain the next state s_{t+1} according to the action a_t. Specifically, the task-caching decisions of BSs can be directly achieved from $\mathbf{Y}(t+1)$ in a_t. Therefore, the total return of minimizing long-term local energy consumption within T timeslots can be given by

$$R = \sum\nolimits_{t\in\mathcal{T}} \gamma w_t, \tag{13}$$

where γ is the reward discount factor satisfying $\gamma \in (0,1)$.

4.2. TD3O Algorithm

The TD3 algorithm is an actor-critic-based framework that comprises the policy (μ) network, critic (Q) network and their corresponding target networks and updates the network parameters using gradient algorithms. It is characterized by using two critic networks and two critic target networks in the design of critic networks. The TD3 algorithm is often divided into two parts consisting of experience collection and training. In the phase

of collecting experience, a new action a_t can be generated by adding random Gaussian noise into the output of the policy network at the state s_t, i.e.,

$$a_t = \mu(s_t, \theta^\mu) + \bar{\sigma}^2. \tag{14}$$

where θ^μ is the parameter of the policy network and $\bar{\sigma}^2$ is the additive Gaussian noise.

After that, the environment is rewarded with w_t and the next state s_{t+1} can be achieved according to the state and action (s_t, a_t). To enable the algorithm to obtain better decisions through past experience-assisted training, we put the quadruple (s_t, a_t, w_t, s_{t+1}) into the experience replay buffer as a historical experience. In the training process, a certain number of quadruples are randomly selected from the experience replay buffer for training. Since the TD3 algorithm consists of policy and critic networks, the training part of the network is relatively independent, so it is divided into the following two parts.

4.2.1. Training Policy Network

The training process of the policy network is shown in Figure 3. In the training phase, N quadruples are extracted from the experience replay buffer and denoted as $\mathcal{N} = \{1, 2, \cdots, N\}$. For any quadruple $n \in \mathcal{N}$, the policy network outputs a new action $a'_n = \mu(s_n, \theta^\mu)$ according to the state s_n. It should be noted that the policy a'_n is different from a_n existing in the experience replay buffer. After s_n and a'_n are inputted into any critic network (e.g., critic Q_1 network), such network outputs $q_n = Q_1(s_n, \mu(s_n, \theta^\mu), \theta^{Q_1})$, where θ^{Q_1} is the parameter of the critic Q_1 network. After achieving all q_n, their mathematical expectation is given by

$$J(\theta^\mu) = \mathbb{E}\Big[Q_1\Big(\mathcal{S}, \mu(\mathcal{S}, \theta^\mu), \theta^{Q_1}\Big)\Big], \tag{15}$$

where $\mathcal{S} = \{s_n | n \in \mathcal{N}\}$. Then, the policy gradient of function J with respect to θ^μ can be given by

$$\nabla_{\theta^\mu} J = \mathbb{E}\Big[\nabla_{\mathcal{A}} Q_1\Big(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\Big) \nabla_{\theta^\mu} \mu(\mathcal{S}, \theta^\mu)\Big], \tag{16}$$

where $\mathcal{A} = \{a_n | n \in \mathcal{N}\}$.

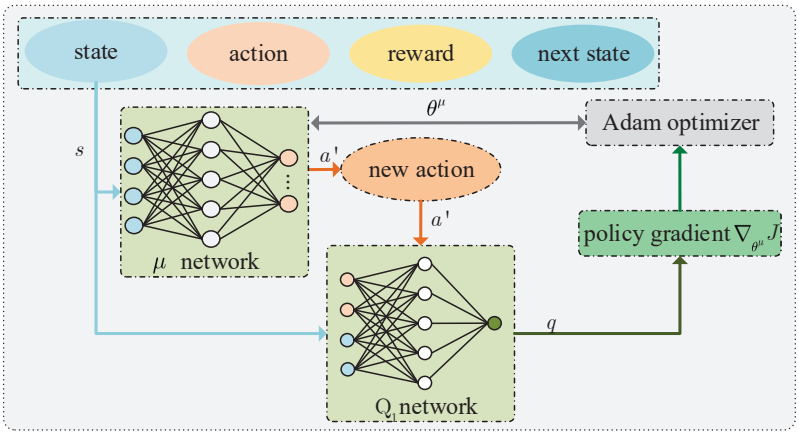

Figure 3. Training policy network.

Significantly, the calculated gradient requires gradient clipping, which can avoid skipping the optimal solution because the gradient is too large. The calculated policy gradients will be used to update the parameters of the policy networks. We assume that the learning rate of the policy network is β^μ, and use the adaptive moment (Adam) estimation commonly used in DRL to obtain the optimal θ^μ [43] .

4.2.2. Training Critic Network

Figure 4 shows the training process of the critic network. During the critic network training, the policy at the next time is first estimated through the state at the next time by

the policy target(μ^-) network, i.e., ${a'}_n = \mu^-({s'}_n, \theta^{\mu^-}) + \hat{\sigma}^2$, where $\hat{\sigma}^2$ is policy noise, that is, trimmed additive Gaussian noise. Then, the action ${a'}_n$ and the state ${s'}_n$ are used as the input of the critic target (Q_1^-) network and critic target (Q_2^-) network, where $\theta^{Q_1^-}$ and $\theta^{Q_2^-}$ are their parameters. After that, these two networks output $\tilde{q}_{n,1}$ and $\tilde{q}_{n,2}$, respectively. Next, the approximation of Q value is $\bar{q}_n = r_n + \gamma\tilde{q}_n$ achieved using Behrman equation, where $\tilde{q}_n = \min(\tilde{q}_{n,1}, \tilde{q}_{n,2})$. At the same time, the action a_n and the state s_n are used as the input of the critic Q_1 network and critic Q_2 network, where θ^{Q_1} and θ^{Q_2} are their parameters. After that, these two networks output $q_{n,1}$ and $q_{n,2}$. At last, for all $\bar{q}_n$, according to the theorem of mean squared error (MSE), the expectation function of the squared loss between $Q_1(\mathcal{S}, \mathcal{A}, \theta^{Q_1})$ and $\bar{Q}$ is

$$L_1\left(\theta^{Q_1}\right) = 0.5\mathbb{E}\left[\left(Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right) - \bar{Q}\right)^2\right], \tag{17}$$

and the expectation function of the squared loss between $Q_2(\mathcal{S}, \mathcal{A}, \theta^{Q_2})$ and $\bar{Q}$ is given by

$$L_2\left(\theta^{Q_2}\right) = 0.5\mathbb{E}\left[\left(Q_2\left(\mathcal{S}, \mathcal{A}, \theta^{Q_2}\right) - \bar{Q}\right)^2\right], \tag{18}$$

where $\bar{Q} = \{\bar{q}_n | n \in \mathcal{N}\}$. Then, the gradient of the loss function $L_1(\theta^{Q_1})$ with respect to the parameter θ^{Q_1} is

$$\nabla_{\theta^{Q_1}} L_1 = \mathbb{E}\left[\left(Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right) - \bar{Q}\right)\nabla_{\theta^{Q_1}} Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right)\right], \tag{19}$$

and the gradient of the loss function $L_2(\theta^{Q_2})$ with respect to the parameter θ^{Q_2} is given by

$$\nabla_{\theta^{Q_2}} L_2 = \mathbb{E}\left[\left(Q_2\left(\mathcal{S}, \mathcal{A}, \theta^{Q_2}\right) - \bar{Q}\right)\nabla_{\theta^{Q_2}} Q_2\left(\mathcal{S}, \mathcal{A}, \theta^{Q_2}\right)\right]. \tag{20}$$

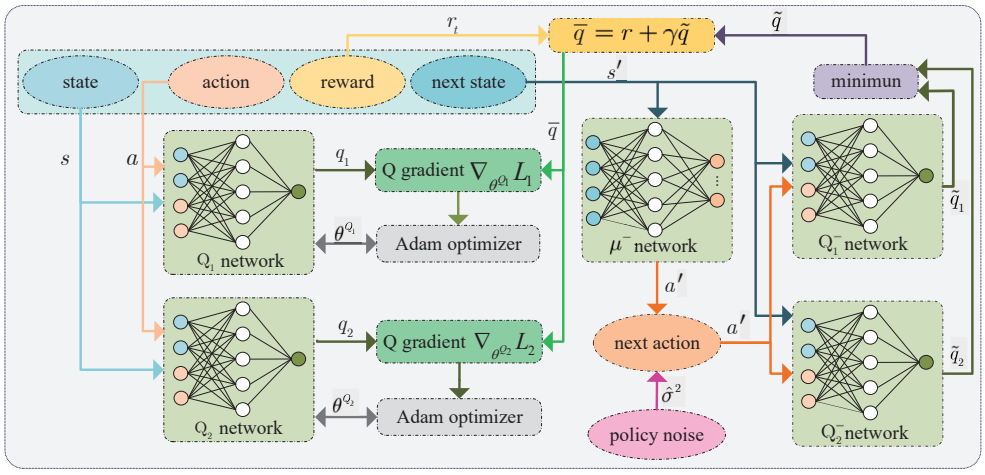

Figure 4. Training critic network.

Similar to calculating the policy gradient, the gradient clipping needs to be performed after calculating the gradients using (19) and (20). In addition, β^Q is the learning rate of the critic network, and the parameters of the two critic networks are updated using the Adam algorithm. Certainly, the parameters of critic target networks also need to be updated using the soft update method, i.e.,

$$\theta^{\mu^-} = \lambda\theta^{\mu} + (1-\lambda)\theta^{\mu^-}, \tag{21}$$

$$\theta^{Q_1^-} = \lambda\theta^{Q_1} + (1-\lambda)\theta^{Q_1^-}, \tag{22}$$

$$\theta^{Q_2^-} = \lambda\theta^{Q_2} + (1-\lambda)\theta^{Q_2^-}, \tag{23}$$

where λ is the learning rate of target networks.

It is noteworthy that a lower network updating frequency is adopted in this paper. We assume that the update interval of the critic network is t^{cti} and the update interval between the policy and critic networks is t^{pti}. The critic networks are trained many times to ensure the stability of Q value. After that, the policy network can be updated. The detailed procedure of the TD3O algorithm is summarized in Algorithm 1, where t^{mep} is the maximal number of epochs.

Algorithm 1: TD3-based offloading (TD3O)

Initialization: θ^{Q_1}, θ^{Q_2}, θ^{μ}, $\theta^{Q_1^-}$, $\theta^{Q_2^-}$, θ^{μ^-}, $t^{\text{step}} = 0$, $t^{\text{epoch}} = 0$.
While $t^{\text{epoch}} < t^{\text{mep}}$
  Let $t = 0$, state s_t and reward $R = 0$.
  While $t < T$
    Generate action a_t using (14).
    Achieve actual action by executing Algorithm 2.
    Calculate reward w_t using (12) and obtain the state s_{t+1}.
    If $t^{\text{step}} \geq \kappa$
      Replace the previous quadruple with (s_t, a_t, w_t, s_{t+1}).
    Else
      Put the quadruple (s_t, a_t, w_t, s_{t+1}) into the queue.
    EndIf
    Update state $s_t = s_{t+1}$.
    If $t^{\text{step}} \% t^{\text{cti}} = 0$ and $t^{\text{step}} > N$
      Extract N quadruples for training.
      For any sample n, Q_1^- and Q_2^- networks output $\tilde{q}_{n,1}$ and
       $\tilde{q}_{n,2}$, respectively, and obtain the minimum value $\tilde{q}_n$.
      Calculate $L(\theta_1^Q)$ and $L(\theta_2^Q)$ using (17) and (18), respectively.
      Calculate Q gradient using (19) and (20), and clip it.
      Find θ^{Q_1} and θ^{Q_2} using Adam optimizer.
      If $t^{\text{step}} \% t^{\text{pti}} = 0$
        Calculate q through Q_1.
        Calculate policy gradient using (16), and clip it.
        Find θ^{μ} using Adam optimizer.
      EndIf
      Calculate $\theta^{Q_1^-}$, $\theta^{Q_2^-}$ and θ^{μ^-} using (21)–(23), respectively.
    EndIf
    $R = R + \gamma w_t$.
    $t^{\text{step}} = t^{\text{step}} + 1$; $t = t + 1$.
  EndWhile
  $t^{\text{epoch}} = t^{\text{epoch}} + 1$.
EndWhile

4.3. AT Algorithm

In order to apply the TD3O algorithm to solve the problem P1, it is necessary to convert the achieved continuous action $a_t = \{\bar{\mathbf{X}}(t), \bar{\mathbf{Y}}(t+1), \bar{\mathbf{Z}}(t), \bar{\mathbf{U}}(t), \bar{\mathbf{F}}(t)\}$ into a discrete one [44]. To this end, we consider the following transformations for a_t.

4.3.1. The Discretization of Device Association Array

In $\bar{\mathbf{X}} = \{\bar{x}_m | \forall m \in \mathcal{M}\}$, $\bar{x}_m$ is the non-integer association index of MD m, which is the continuous action achieved by the TD3 algorithm. Then, it is converted into an integer form, i.e.,

$$\begin{cases} x_{m,\text{ceil}(I\bar{x}_m)} = 1, \text{ if } I\bar{x}_m \neq 0, \\ x_{m,1} = 1, \text{ otherwise}, \end{cases} \tag{24}$$

where $\text{ceil}(b)$ is an upward rounding function with respect to b. Such a transformation can ensure that each MD can be associated with one BS.

Algorithm 2: Action transformation (AT)

For each MD $m \in \mathcal{M}$
  Achieve MD association matrix $\mathbf{X}$ using discretization rule.
  Achieve task caching matrix $\mathbf{Y}$ using discretization rule.
  Achieve task offloading matrix $\mathbf{U}$ using discretization rule.
EndFor
For each BS $i \in \mathcal{I}$
  Returns the set $\mathcal{K}_i$ of available subchannels and the set $\mathcal{M}_i$ of
  offloading MDs.
  If $M_i > K_i$
    $M_i - K_i$ associated MDs are randomly selected, disassociated
    and execute tasks locally.
  EndIf
  Achieve subchannel allocation matrix $\mathbf{Z}$ using discretization rule.
EndFor
For each MD $m \in \mathcal{M}$
  If $\sum_{i \in \mathcal{I}} u_{m,i} = 1$
    If $\tilde{f}_m = 0$
      Assign small enough computing capacity to MD m to avoid
      zero division.
    Else
      Allocate computing resources to MD m using (28).
    EndIf
  EndIf
EndFor

4.3.2. The Discretization of Task-Caching Array

In $\bar{\mathbf{Y}}$, $\bar{y}_m$ represents the non-integer caching index of MD m, which is the continuous action achieved by the TD3 algorithm. Since each MD can store its task at all BSs, there exist 2^I storage options for it. Consequently, in order to convert $\bar{y}_m$ into a discrete form, we first need to perform

$$\begin{cases} \hat{y}_m = \text{floor}\left(2^I \bar{y}_m\right), \text{ if } 2^I \bar{y}_m \neq 0, \\ \hat{y}_m = 0, \text{ otherwise}, \end{cases} \tag{25}$$

where $\text{floor}(b)$ is a downward rounding function with respect to b. Then, in order to achieve the binary caching index, the decimal $\hat{y}_m$ needs to be converted into a binary number of I 0–1 digits, which is given by $\text{bin}(\bar{y}_m)$. In it, $\text{bin}(b)$ is a function used for calculating the binary number of decimal b. Then, $y_{m,i} = \text{bin}(\bar{y}_m)_i$, where $\text{bin}(\bar{y}_m)_i$ represents the i-th digit of the binary number $\text{bin}(\bar{y}_m)$.

4.3.3. The Discretization of Task Offloading Array

In $\bar{\mathbf{U}}$, $\bar{u}_m$ is the non-integer offloading index of MD m, which is the continuous action achieved by the TD3 algorithm. Considering that each MD can offload its task to at most one BS, $\bar{u}_m$ is converted into an integer form, i.e.,

$$\begin{cases} u_{m,\text{ceil}(I\bar{u}_m)} = 1, \text{ if } I\bar{u}_m \neq 0, \\ u_{m,i} = 0, \forall i \in \mathcal{I}, \text{ otherwise}. \end{cases} \tag{26}$$

4.3.4. The Discretization of Subchannel Allocation Array

In $\bar{\mathbf{Z}}$, $\bar{z}_i$ is the non-integer index of the subchannels allocated by BS i to its associated MDs who need to offload tasks, which is the continuous action achieved by the TD3 algorithm. To achieve the integer form of $\bar{z}_i$, we first need to perform

$$\begin{cases} \hat{z}_i = \text{ceil}(\text{C}(M_i, K_i)\bar{z}_i), \text{ if } \text{C}(M_i, K_i)\bar{z}_i \neq 0, \\ \hat{z}_i = 1, \text{ otherwise}, \end{cases} \tag{27}$$

where M_i is the number of MDs that are associated with BS i and need to offload tasks; K_i is the number of available subchannels at BS i; $\mathrm{C}(M_i, K_i) = \mathrm{fac}(K_i)/\mathrm{fac}(M_i)\mathrm{fac}(K_i - M_i)$ is a function with respect to M_i and K_i and is used for calculating the number of feasible subchannel allocation policies between M_i MDs and K_i subchannels at BS i; $\mathrm{fac}(b)$ is a factorial function with respect to b; $M_i \leq K_i$ shall be satisfied.

Then, we assume that $\mathcal{Z}_i = \{1, 2, \cdots, \mathrm{C}(M_i, K_i)\}$ is the set of $\mathrm{C}(M_i, K_i)$ feasible subchannel allocation policies between M_i MDs and K_i subchannels at BS i. After that, the subchannel allocation policy $\hat{z}_i$ in the set $\mathcal{Z}_i$ is selected according to the Equation (27). It is noteworthy that $\mathrm{C}(M_i, K_i)$ feasible subchannel allocation policies are generated in advance. That is to say, in the policy $\hat{z}_i$, we can easily know the utilized indices of K_i subchannels for M_i MDs. According to these rules, we can easily find the subchannel allocation index $\mathbf{Z}$.

4.3.5. The Transformation of Computing Resource Allocation Array

In $\bar{\mathbf{F}} = \left\{\bar{f}_m | \forall m \in M\right\}$, $\bar{f}_m$ represents the computing resource score of MD m at the target BS that is executing its task. If $\sum_{i \in I} u_{m,i} = 1$ is satisfied between MD m and BS i, according to the proportional allocation of computing resources, the computing resources allocated to MD m by BS i can be given by

$$f_{m,i} = u_{m,i} f_i^{\mathrm{BS}} \bar{f}_m / \sum_{j \in \mathcal{M}} u_{j,i} \bar{f}_j. \tag{28}$$

Based on the above-mentioned operations, the output action $a_t = \left\{\bar{\mathbf{X}}, \bar{\mathbf{Y}}, \bar{\mathbf{Z}}, \bar{\mathbf{U}}, \bar{\mathbf{F}}\right\}$ of the TD3O algorithm can be effectively converted into an actual decision, which is summarized as Algorithm 2.

4.4. HA

To solve the problem P1 in a non-iterative manner, we design an effective heuristic algorithm, which is summarized in Algorithm 3. In such an algorithm, to reduce the uplink transmission time and energy consumption, some MDs are associated with the nearest BSs, and the BSs randomly cache the tasks of their associated MDs until the cache space cannot cache more tasks. Then, the uncached MDs randomly select a BS as the offloading target, and to guarantee time constraints, the BS will evenly distribute the computing resources according to the computation amount of the task. Finally, a part of the MDs are disassociated from BSs without sufficient subchannels and execute tasks by themselves.

Algorithm 3: Heuristic algorithm (HA)

```
Initialization: energy consumption ε̄^tot = 0.
Each MD selects (is associated with) the nearest BS.
For each BS i ∈ I
  If M_i > K_i
    M_i − K_i associated MDs are randomly selected, disassociated
      and execute tasks locally.
  EndIf
  Randomly select the tasks of MDs associated with BS i for caching
   until the caching space is full.
EndFor
For t ∈ T
  Randomly select a target BS for each MD without cached task.
  Randomly allocate subchannels to MDs associated with each BS.
  If subchannels are insufficient
    Extra MDs are randomly selected to execute tasks locally.
  EndIf
  Proportionally allocate computing resources to MDs associated with
   each BS according to the CPU cycles required by tasks.
  Calculate the total local energy consumption ε̄.
  ε̄^tot = ε̄^tot + ε̄.
EndFor
```

5. Algorithm Analysis

5.1. Computation Complexity Analysis

In this section, the computation complexity of proposed algorithms are analysed as follows.

Proposition 1. *The computation complexity of Algorithm 2 is $\mathcal{O}(MIK)$ in the worst case.*

Proof. In Algorithm 2, the computation complexity of steps 1–5 is $\mathcal{O}(M)$, the computation complexity of steps 6–14 is $\mathcal{O}(MIK)$ in the worst case, and the computation complexity of steps 15–24 is $\mathcal{O}(MI)$. In general, the computation complexity of Algorithm 2 is $\mathcal{O}(MIK)$ in the worst case. □

Proposition 2. *The computation complexity of Algorithm 1 is $\mathcal{O}\big(\max\big(\sum_{l=0}^{L^Q}\psi_l^Q\psi_{l+1}^Q, \sum_{l=0}^{L^\mu}\psi_l^\mu\psi_{l+1}^\mu\big)\big)$ at each timeslot, where L^μ is the number of layers of the policy network, L^Q is the number of layers of the critic network, ψ_l^μ is the number of neurons at the l-th layer of the policy network and ψ_l^Q is the number of neurons at l-th layer of the critic network.*

Proof. In Algorithm 1, the computation complexity is mainly related to the action transformation, the calculation of the reward and task processing time and the structure of the neural network. As previously mentioned, the computation complexity of the action transformation should be $\mathcal{O}(MIK)$ in the worst case. As seen from Formulas (8) and (12), the computation complexity of the calculation of the reward and task processing time is $\mathcal{O}(MIK)$.

In Algorithm 1, there exist four critic networks and two policy networks. We assume that the structure of the policy network and its target network is the same, and the structure of the two critic networks and its target network is the same. Then, we can easily deduce that the computation complexity of establishing policy networks is $\mathcal{O}\big(\sum_{l=0}^{L^\mu}\psi_l^\mu\psi_{l+1}^\mu\big)$ and the computation complexity of establishing critic networks is $\mathcal{O}\big(\sum_{l=0}^{L^Q}\psi_l^Q\psi_{l+1}^Q\big)$. Therefore, the computation complexity of establishing neural networks is $\mathcal{O}\big(\max\big(\sum_{l=0}^{L^Q}\psi_l^Q\psi_{l+1}^Q, \sum_{l=0}^{L^\mu}\psi_l^\mu\,\psi_{l+1}^\mu\big)\big)$.

Since the computation complexity of establishing neural networks is much higher than that of the other operations in Algorithm 2. In general, the computation complexity of Algorithm 2 is $\mathcal{O}\big(\max\big(\sum_{l=0}^{L^Q}\psi_l^Q\psi_{l+1}^Q, \sum_{l=0}^{L^\mu}\psi_l^\mu\psi_{l+1}^\mu\big)\big)$ at each timeslot. □

Proposition 3. *The computation complexity of Algorithm 3 is $\mathcal{O}(MI)$ at each timeslot.*

Proof. In Algorithm 3, the computation complexity of step 2 is $\mathcal{O}(MI)$, the computation complexity of steps 3–10 is $\mathcal{O}(I)$, the computation complexity of steps 12–16 is $\mathcal{O}(M)$, the computation complexity of steps 17–19 is $\mathcal{O}(MI)$. In general, the computation complexity of Algorithm 3 is $\mathcal{O}(MI)$ at each timeslot. □

5.2. Convergence Analysis

Since Algorithm 2 is a part of Algorithms 1 and 3 and is non-iterative, we just need to concentrate on the convergence of Algorithm 1. In detail, this is established as follows.

Theorem 1. *Algorithm 1 can be guaranteed to converge after finite iterations.*

Proof. In Algorithm 1, the neural networks are updated by the gradient descent method used in the Adam optimizer. This utilizes the gradient information of the functions $J(\theta^\mu)$, $L_1(\theta^{Q_1})$ and $L_2(\theta^{Q_2})$ to guide the updating directions of the parameters θ^μ, θ^{Q_1} and θ^{Q_2}, so that the objective functions can reach the optimal or suboptimal values. When these values tend to be stable, the parameters θ^μ, θ^{Q_1} and θ^{Q_2} also tend to be stable. At this time, Algorithm 1 is deemed convergent. □

6. Performance Evaluation

In order to verify the performance of the designed algorithms, we introduce the following algorithms for comparison.

DDPG-based offloading (DDPGO): DDPG is a classical DRL algorithm [45]. Compared with the TD3 algorithm, the DDPG algorithm reduces the critic network and the critic target network. In addition, both the critic network and policy network are updated at each timeslot in the DDPG algorithm. In this paper, the DDPG algorithm used to solve the problem P1 is named the DDPG-based offloading (DDPGO) algorithm. The difference between the two algorithms is that the state input and action output are the same as the mode used by TD3O, and it also uses the AT algorithm to convert continuous actions.

Completely offloading (CO): In the CO algorithm, the task of each MD is offloaded to the nearest BS for computing. Such BS proportionally allocates the computing capacity to its associated MDs according to the CPU cycles required by the tasks of these MDs.

Completely local executing (CLE): In the CLE algorithm, the tasks of all MDs can be executed by themselves.

In this paper, we consider that each BS is deployed in a non-overlapping area with a radius of 400 m and a power spectral density of -174 dBm/Hz. In addition, $I = 3$, $f_m^{\mathrm{loc}} = 1$ GHz, $f_i = 8$ GHz, $W = 40$ MHz, $K = 4$, $d_m = 2{\sim}5$ MB, $c_m = 50$ cycles/bit, $\xi = 10^{-27}$, $\tau_m = 10$ s, $\rho = 2$, $r^{\mathrm{bh}} = 1$ Gbps, $p_m = 23$ dBm, $\kappa = 80{,}000$, $N = 128$, $\gamma = 0.94$ and $\lambda = 0.04$. In the DRL algorithm, we consider that both the policy network and the critic network are composed of three-layer fully connected neural networks, where the numbers of neurons in three-layer neural networks in the policy network are 300, 200 and 128, respectively, and the corresponding target network has the same structure with this policy network; the number of neurons in three-layer neural networks in the critic network are 300, 128 and 32, respectively, and the corresponding target network has the same structure as this critic network. Significantly, the first-layer fully connected neural network of the policy network and the critic network utilizes the rectified linear unit 6 (RELU6), which suppresses the maximum value as the activation function, while other layers use RELU as the activation function.

Figure 5 shows the convergence of the TD3O and DDPGO algorithms. As shown in Figure 5, DDPGO may have a higher convergence rate than TD3O, but the former may have worse convergence stability than the latter. The reason for this may be that the critic network and the policy network are updated synchronously in DDPGO. In DDPGO, the network parameters are updated in each training phase, which speeds up the convergence. Synchronously, the policy network parameters are updated in the training, which results in the instability of the long-term reward value and training bias. As we know, TD3O is composed of two sets of critic networks. Consequently, it could be trained in a relatively stable Q value so that the algorithm can converge stably. In the simulation, it is also easy to find that TD3O could achieve a more stable and better solution to the problem P1 than DDPGO in general.

Figure 6 shows the impact of the training interval t^{pti} on the convergence of the TD3O algorithm. As we know, under the same number of iterations, a larger t^{pti} can effectively reduce the overall training time of the network. However, it will reduce the total learning times of the policy network and its target network. As illustrated in Figure 6, the convergence rate of TD3O may decrease with t^{pti} in general.

Figure 7 shows the impacts of learning rates β^Q and β^μ on the convergence of the TD3O algorithm. As we know, when the learning rate β^Q of the critic network increases, the parameters of such network will be updated at a larger scale, which speeds up the convergence of TD3O. However, it may lead to the failure of stable evaluation of environmental information, which weakens the convergence stability of TD3O. As illustrated in Figure 7, when $\beta^Q = 0.001$, the convergence rate of TD3O is relatively high, but the achieved long-term reward dramatically fluctuates at this moment. On the other hand, the learning rate β^μ of the policy network can affect the optimization capability of TD3O.

Specifically, a lower β^{μ} means a smaller amplitude of updating the policy network, which is better for finding better solutions. As seen from Figure 7, TD3O can achieve a better long-term reward when $\beta^{Q} = 0.0001$ and $\beta^{\mu} = 0.0001$.

Figure 8 shows the impact of the number of MDs on the long-term local energy consumption $\varepsilon^{\mathrm{MD}}$, where $\varepsilon^{\mathrm{MD}}$ is the sum of the total local energy consumption in T timeslots. In general, the $\varepsilon^{\mathrm{MD}}$ increases with the number of MDs since a greater energy consumption is used when tackling more tasks of more MDs. Since CLE executes tasks in maximal computation capacity, it could achieve the highest $\varepsilon^{\mathrm{MD}}$ among all algorithms. In CO, MDs are associated with the nearest BSs, which may result in a relatively imbalanced load distribution. Then, some overloaded BSs cannot provide good services for their associated MDs because of limited resources, which may result in high $\varepsilon^{\mathrm{MD}}$. Consequently, CO could achieve higher $\varepsilon^{\mathrm{MD}}$ than other algorithms excluding CLE. As illustrated in Figure 8, TD3O could achieve lower $\varepsilon^{\mathrm{MD}}$ than DDPGO since the former can effectively mitigate the overestimation existing in the latter. Although HA lets MDs be associated with the nearest BSs, some MDs associated with overloaded BSs will disassociate and execute tasks locally. Such an operation may result in relatively low $\varepsilon^{\mathrm{MD}}$.

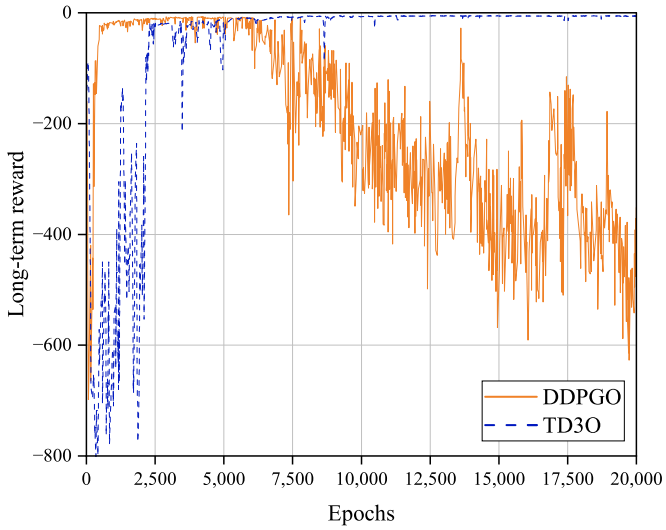

Figure 5. The convergence of DDPGO and TD3O algorithms.

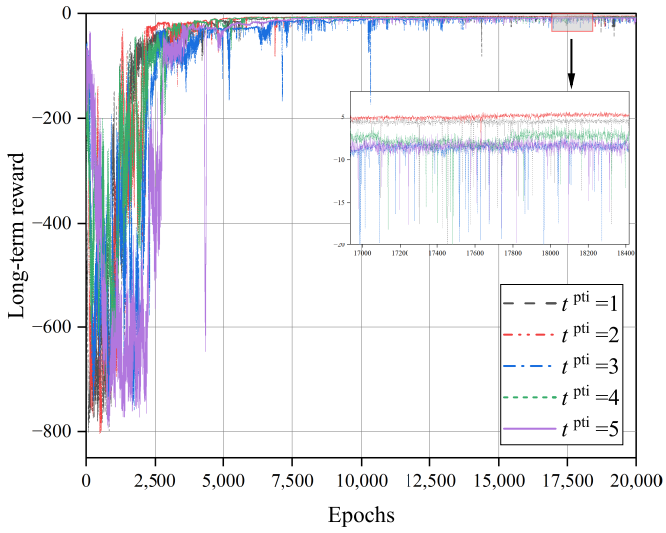

Figure 6. The impact of training interval t^{pti} on the convergence of TD3O algorithm.

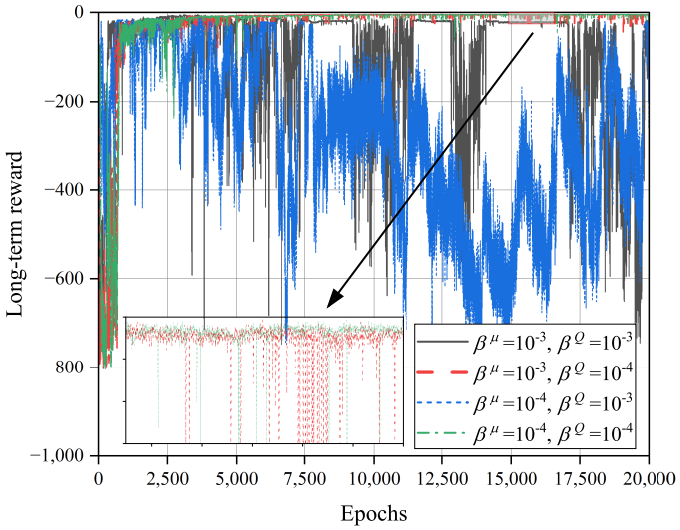

Figure 7. The impacts of learning rates β^{Q} and β^{μ} on the convergence of TD3O algorithm.

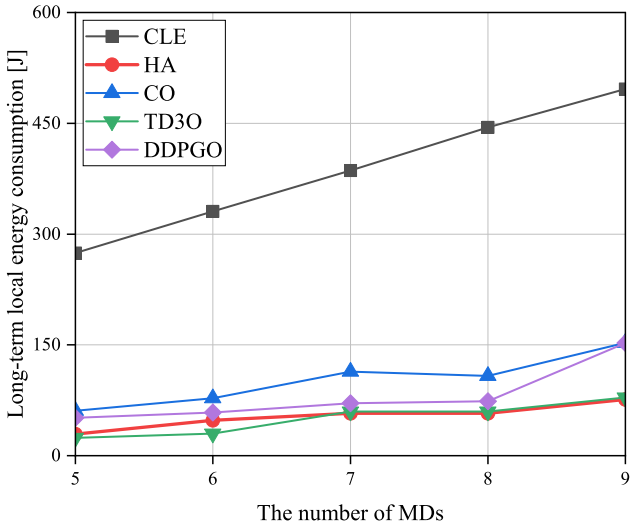

Figure 8. The impacts of the number of MDs on the long-term local energy consumption ε^{MD}.

Figure 9 shows the impacts of the number of MDs on the long-term reward (R). As illustrated in Figure 9, R may decrease with the number of MDs since more MDs result in a higher energy consumption. Since both TD3O and DDPGO try to maximize the reward but in other algorithms this is not the case, the former could achieve higher R than other algorithms in general. Since TD3O could achieve lower ε^{MD} than DDPGO, the former could achieve a higher R than the latter. In view of the unstable convergence of DDPGO, its reward may dramatically fluctuate. Since CLE could achieve the highest ε^{MD} among all algorithms, it could achieve the lowest R in general. In addition, CO could achieve a lower R than HA since the former consumes more energy than the latter.

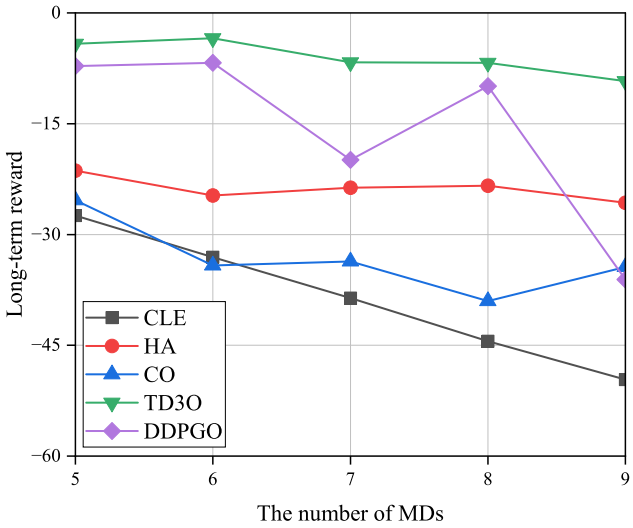

Figure 9. The impacts of the number of MDs on the long-term reward R.

Figure 10 shows the impacts of the size of caching space of each BS on the relative long-term energy consumption η^{ε} denoted as the ratio of $\varepsilon^{\mathrm{MD}}$ achieved by an offloading algorithm to the one attained by CLE. Evidently, a smaller η^{ε} means a higher energy gain caused by offloading tasks. As illustrated in Figure 10, in addition to DDPGO and CO, η^{ε} in other algorithms decreases with the size of the caching space of each BS in general. The reason for this may be that a larger caching space can hold more tasks to reduce the transmission energy consumption. However, as revealed in Figures 5 and 9, the unstable convergence of DDPGO may result in a dramatically fluctuating performance. Therefore, η^{ε} in DDPGO may evidently be fluctuating. In addition, η^{ε} in CO may not change with the size of caching space of each BS since it does not utilize caching space. By minimizing the $\varepsilon^{\mathrm{MD}}$, TD3O and DDPGO could achieve a lower η^{ε} than other algorithms in general. In addition, TD3O could achieve a lower η^{ε} than DDPGO since the former mitigates the overestimation existing in DDPGO. As seen from Figure 10, CO could achieve the highest η^{ε} among all algorithms since it has no sufficient resources to provide for MDs associated with overloaded BSs.

Figure 11 shows the impacts of the size of the caching space of each BS on the relative long-term reward η^{R} denoted as the ratio of the long-term reward achieved by an offloading algorithm to the one attained by CLE. Evidently, a smaller η^{R} means a higher reward gain caused by offloading tasks. As illustrated in Figure 11, η^{R} in TD3O and HA decreases with the size of the caching space of each BS in general. The reason for this may be that a larger caching space can hold more tasks to reduce the transmission energy consumption, and then bring a higher reward. However, due to the unstable convergence of DDPGO, η^{R} in DDPGO may be fluctuating. Moreover, η^{R} in CO may not change with the size of caching space of each BS since it does not utilize caching space. By minimizing the $\varepsilon^{\mathrm{MD}}$ and thus increasing the reward, TD3O and DDPGO could achieve a lower η^{R} than other algorithms in general. In addition, TD3O could achieve a lower η^{R} than DDPGO since the former mitigates the overestimation existing in DDPGO. As can be seen from Figure 11, CO could achieve the highest η^{R} because of the high energy consumed by MDs associated with overloaded BSs.

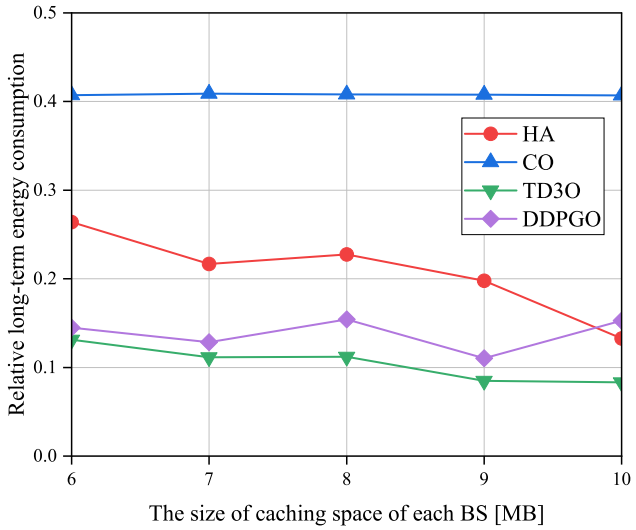

Figure 10. The impacts of the size of caching space of each BS on the relative long-term energy consumption η^{ε}.

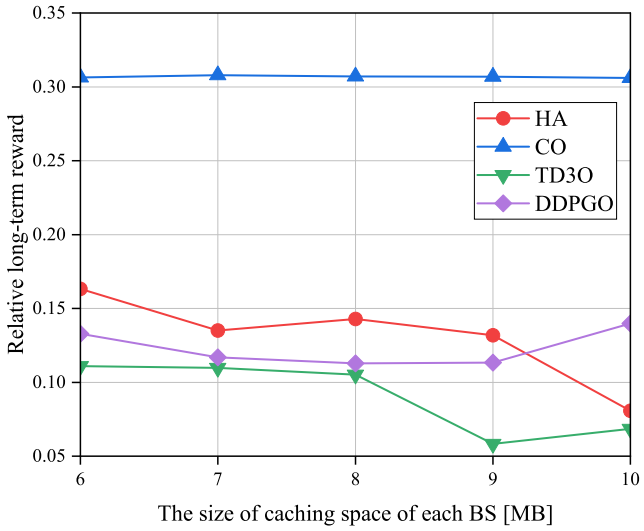

Figure 11. The impacts of the size of caching space of each BS on the relative long-term reward η^{R}.

As seen from the above-mentioned simulation figures, although HA is a non-iterative algorithm, it could sometimes achieve better performance than DDPGO. In addition, it may always achieve fairly better performance than CO and CLE.

7. Conclusions

In this paper, the problem of minimizing the local energy consumption is concentrated in the cache-assisted vehicular NOMA–MEC networks under time and resource constraints, which refers to the joint optimization of the computing resource allocation, subchannel selection, device association, offloading and caching decisions. To solve the formulated problem, we developed an effective TD3O algorithm that was integrated with the AT algorithm and designed HA simultaneously. As for the designed algorithms, we have given some analyses of the convergence and computation complexity. Simulation results show that TD3O could achieve lower local energy consumption than several benchmark algorithms, and HA could achieve lower local energy consumption than the CO and CLE algorithms. Future work can include power allocation and secure communications, such as

optimizing the transmission power of MDs for task offloading, and how to encrypt part of the task data in the network at a low cost to achieve secure communication, etc.

Author Contributions: Conceptualization, T.Z. and M.X.; methodology, T.Z. and M.X.; software, M.X.; investigation, X.N., X.L. and C.L.; writing—original draft preparation, T.Z. and M.X.; writing—review and editing, D.Q. 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 under grant Nos. 62261020, 62171119, 62062034, 62361026, 61961020, 62001201 and 61963017; the National Key Research and Development Program of China under grant No. 2020YFB1807201; the Jiangxi Provincial Natural Science Foundation under grant Nos. 20232ACB212005, 20224BAB202001, 20232BAB202019, 20212BAB202004 and 20212BAB212001; the key research and development plan of Jiangsu Province under grant No. BE2021013-3.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

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

References

1. Yao, Y.; Shu, F.; Li, Z.; Cheng X.; Wu, L. Secure transmission scheme based on joint radar and communication in mobile vehicular networks. *IEEE Trans. Intell. Transport. Syst.* **2023**, *24*, 10027–10037. [CrossRef]
2. Yao, Y.; Zhao, J.; Li, Z.; Cheng X.; Wu, L. Jamming and eavesdropping defense scheme based on deep reinforcement learning in autonomous vehicle networks. *IEEE Trans. Inf. Forens. Secur.* **2023**, *18*, 1211–1224. [CrossRef]
3. Wu, Q.; Shi, S.; Wan, Z.; Fan, Q.; Fan, P.; Zhang, C. Towards V2I age-aware fairness access: A dqn based intelligent vehicular node training and test method. *Chin. J. Electron.* **2023**, *32*, 1230–1244.
4. Khan, S.; Luo, F.; Zhang, Z.; Ullah, F.; Amin, F.; Qadri, S.; Heyat, M.; Ruby, R.; Wang, L.; Ullah, S.; et al. A survey on X.509 public-key infrastructure, certificate revocation, and their modern implementation on blockchain and ledger technologies. *IEEE Commun. Surv. Tutor.* 2023; *early access*. [CrossRef]
5. Khan, S.; Luo, F.; Zhang, Z.; Rahim, M.; Ahmad, M.; Wu, K. Survey on issues and recent advances in vehicular public-key infrastructure (VPKI). *IEEE Commun. Surv. Tutor.* **2023**, *24*, 1574–1601. [CrossRef]
6. Wang, D.; Tian, X.; Cui, H.; Liu, Z. Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware MEC network. *China Commun.* **2020**, *17*, 31–44. [CrossRef]
7. Spinelli, F.; Mancuso, V. Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 596–630. [CrossRef]
8. Zhang, Y.; Di, B.; Zheng, Z.; Lin, J.; Song, L. Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 2565–2578. [CrossRef]
9. Zhou, T.; Yue, Y.; Qin, D.; Nie, X.; Li, X.; Li, C. Joint device association, resource allocation, and computation offloading in ultradense multidevice and multitask IoT networks. *IEEE Internet Things J.* **2022**, *9*, 18695–18709. [CrossRef]
10. Zhang, W.; Zhang, G.; Mao, S. Joint parallel offloading and load balancing for cooperative-MEC systems with delay constraints. *IEEE Trans. Veh. Technol.* **2022**, *71*, 4249–4263. [CrossRef]
11. Malik, R.; Vu, M. Energy-efficient joint wireless charging and computation offloading in MEC systems. *IEEE J. Sel. Top. Signal Proces.* **2021**, *15*, 1110–1126. [CrossRef]
12. Hu, H.; Song, W.; Wang, Q.; Hu, R.Q.; Zhu, H. Energy efficiency and delay tradeoff in an MEC-enabled mobile IoT network. *IEEE Internet Things J.* **2022**, *9*, 15942–15956. [CrossRef]
13. Wu, Q.; Wang, S.; Ge, H.; Fan, P.; Fan, Q.; Letaief, K. Delay-sensitive task offloading in vehicular fog computing-assisted platoons. *IEEE Trans. Netw. Service Manag.* 2023; *early access*. [CrossRef]
14. Bai, Y.; Zhao, H.; Zhang, X.; Chang, Z.; Jäntti, R.; Yang, K. Towards autonomous multiuav wireless network: A survey of reinforcement learning-based approaches. *IEEE Commun. Surv. Tutor.* 2023; *early access*. [CrossRef]
15. Liu, Y.; Liu, J.; Argyriou, A.; Wang, L.; Xu, Z. Rendering-aware VR video caching over multi-cell MEC networks. *IEEE Trans. Veh. Technol.* **2021**, *70*, 2728–2742. [CrossRef]
16. Lekharu, A.; Jain, M.; Sur, A.; Sarkar, A. Deep learning model for content aware caching at MEC servers. *IEEE Trans. Netw. Service Manag.* **2022**, *19*, 1413–1425. [CrossRef]
17. Huang, X.; He, L.; Wang, L.; Li, F. Towards 5G: Joint optimization of video segment caching, transcoding and resource allocation for adaptive video streaming in a multi-access edge computing network. *IEEE Trans. Veh. Technol.* **2021**, *70*, 10909–10924. [CrossRef]
18. Zheng, G.; Xu, C.; Wen, M.; Zhao, X. Service caching based aerial cooperative computing and resource allocation in multi-UAV enabled MEC systems. *IEEE Trans. Veh. Technol.* **2022**, *71*, 10934–10947. [CrossRef]

19. Wu, Q.; Wang, X.; Fan, Q.; Fan, P.; Zhang, C.; Li, Z. High stable and accurate vehicle selection scheme based on federated edge learning in vehicular networks. *China Commun.* **2023**, *20*, 1–17. [CrossRef]
20. Liu, B.; Liu, C.; Peng, M. Resource allocation for energy-efficient MEC in NOMA-enabled massive IoT networks. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 1015–1027. [CrossRef]
21. Song, Z.; Liu, Y.; Sun, X. Joint task offloading and resource allocation for NOMA-enabled multi-access mobile edge computing. *IEEE Trans. Commun.* **2021**, *69*, 1548–1564. [CrossRef]
22. Ding, Z.; Xu, D.; Schober, R.; Poor, H.V. Hybrid NOMA offloading in multi-user MEC networks. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 5377–5391. [CrossRef]
23. Farha, Y.A.; Ismail, M.H. Design and optimization of a UAV-enabled non-orthogonal multiple access edge computing IoT system. *IEEE Access* **2022**, *10*, 117385–117398. [CrossRef]
24. Wang, K.; Li, H.; Ding, Z.; Xiao, P. Reinforcement learning based latency minimization in secure NOMA-MEC systems with hybrid SIC. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 408–422. [CrossRef]
25. Xu, C.; Zheng, G.; Zhao, X. Energy-minimization task offloading and resource allocation for mobile edge computing in NOMA heterogeneous networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 16001–16016. [CrossRef]
26. Ylmaz, S.S.; Zbek, B. Massive MIMO-NOMA based MEC in task offloading for delay minimization. *IEEE Access* **2023**, *11*, 162–170. [CrossRef]
27. Tuong, V.D.; Truong, T.P.; Nguyen, T.-V.; Noh, W.; Cho, S. Partial computation offloading in NOMA-assisted mobile-edge computing systems using deep reinforcement learning. *IEEE Internet Things J.* **2021**, *8*, 13196–13208. [CrossRef]
28. 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]
29. Ren, D.; Gui, X.; Zhang, K. Adaptive request scheduling and service caching for MEC-assisted IoT networks: An online learning approach. *IEEE Internet Things J.* **2022**, *9*, 17372–17386. [CrossRef]
30. Yang, S.; Liu, J.; Zhang, F.; Li, F.; Chen, X.; Fu, X. Caching-enabled computation offloading in multi-region MEC network via deep reinforcement learning. *IEEE Internet Things J.* **2022**, *9*, 21086–21098. [CrossRef]
31. Zhou, T.; Yue, Y.; Qin, D.; Nie, X.; Li, X.; Li, C. Mobile device association and resource allocation in HCNs with mobile edge computing and caching. *IEEE Syst. J.* **2023**, *17*, 976–987. [CrossRef]
32. Wu, Q.; Zhao, Y.; Fan, Q.; Fan, P.; Wang, J.; Zhang, C. Mobility-aware cooperative caching in vehicular edge computing based on asynchronous federated and deep reinforcement learning. *IEEE J. Sel. Top. Signal Process.* **2023**, *17*, 66–81. [CrossRef]
33. 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]
34. Zhang, W.; Zeadally, S.; Zhou, H.; Zhang, H.; Wang, N.; Leung, V. Joint service quality control and resource allocation for service reliability maximization in edge computing. *IEEE Trans. Commun.* **2023**, *71*, 935–948. [CrossRef]
35. Li, X.; Zhang, H.; Zhou, H.; Wang, N.; Long, K.; Al-Rubaye, S.; Karagiannidis, G. Multi-agent drl for resource allocation and cache design in terrestrial-satellite networks. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 5031–5042. [CrossRef]
36. Yang, Z.; Liu, Y.; Chen, Y.; Al-Dhahir, N. Cache-aided NOMA mobile edge computing: A reinforcement learning approach. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 6899–6915. [CrossRef]
37. 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]
38. Huynh, L.N.T.; Pham, Q.-V.; Nguyen, T.D.T.; Hossain, M.D.; Shin, Y.-R.; Huh, E.-N. Joint Computational offloading and data-content caching in NOMA-MEC networks. *IEEE Access* **2021**, *9*, 12943–12954. [CrossRef]
39. Yang, Z.; Ding, Z.; Fan, P.; Al-Dhahir, N. A general power allocation scheme to guarantee quality of service in downlink and uplink NOMA systems. *IEEE Trans. Wirel. Commun.* **2016**, *15*, 7244–7257. [CrossRef]
40. Cheng, Y.; Liang, C.; Chen, Q.; Yu, F.R. Energy-efficient D2D-assisted computation offloading in NOMA-enabled cognitive networks. *IEEE Trans. Veh. Technol.* **2020**, *70*, 13441–13446. [CrossRef]
41. Li, C.; Wang, H.; Song, R. Mobility-aware offloading and resource allocation in NOMA-MEC systems via DC. *IEEE Commun. Lett.* **2022**, *26*, 1091–1095. [CrossRef]
42. Huang, H.; Ye, Q.; Zhou, Y. 6G-empowered offloading for realtime applications in multi-access edge computing. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 1311–1325. [CrossRef]
43. Chen, J.; Xing, H.; Xiao, Z.; Xu, L.; Tao, T. A DRL agent for jointly optimizing computation offloading and resource allocation in MEC. *IEEE Internet Things J.* **2021**, *8*, 17508–17524. [CrossRef]
44. Dai, Y.; Zhang, K.; Maharjan, S.; Zhang, Y. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12175–12186. [CrossRef]
45. Long, D.; Wu, Q.; Fan, Q.; Fan, P.; Li, Z.; Fan, J. A power allocation scheme for MIMO-NOMA and D2D vehicular edge computing based on decentralized DRL. *Sensors* **2023**, *23*, 3449. [CrossRef] [PubMed]