UAV-D2D Assisted Latency Minimization and Load Balancing in Mobile Edge Computing with Deep Reinforcement Learning

Qinglin Song and Long Qu(✉)

Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, Zhejiang, China
{2211100149,qulong}@nbu.edu.cn

Abstract. Now Unmanned Aerial Vehicle (UAV) with Mobile Edge Computing (MEC) severs and Device-to-Device (D2D) communications provide offload computing services for User Devices (UDs). However, the UAV has relatively high transmission latency. And D2D lacks the necessary flexibility. In this paper, we introduce a novel MEC system that utilizes the collaborative advantages of flexible movement of UAV and the low latency transmission of D2D communication to process tasks from UDs. We formulate an optimization problem focused on minimizing the tasks transmission and execution delay of UDs. The problem involves joint optimization of user scheduling, UAV trajectory, and resource allocation of Virtual Machines (VMs) on the MEC server. To tackle this non-convex problem, we propose a Deep Reinforcement Learning (DRL) algorithm with Deep Deterministic Policy Gradient (DDPG). Through simulation results, we demonstrate that DDPG reduces the latency by 41% compared to Deep Q-Network (DQN) and Actor-Critic (AC) algorithm. Our collaborative UAV-D2D model has 16% and 32% lower latency than when only the UAV or D2D works alone.

Keywords: Mobile Edge Computing · Unmanned Aerial Vehicle · Device-to-Device · Virtual Machines · Deep Deterministic Policy Gradient

1 Introduction

In the era of 5G, mobile networks cater to a wide range of devices, including computers, mobile vehicles, and various types of sensors. With the rapid proliferation of Internet of Things (IoT) devices [1], there is a growing demand for applications with stringent requirements for low latency, such as Virtual Reality (VR), Augmented Reality (AR), and video streaming [2]. The traditional core networks are unable to meet the demands of latency-sensitive tasks. To address this challenge, Mobile Edge Computing (MEC) has emerged as a promising solution. MEC aims to reduce processing latency and enhance user experience by

H. Jin et al. (Eds.): GPC 2023, LNCS 14504, pp. 108–122, 2024.
https://doi.org/10.1007/978-981-99-9896-8_8

offloading compute-intensive tasks to edge servers close to the users [3]. Nevertheless, conventional MEC servers are typically deployed on static Base Stations (BSs) and lack the necessary flexibility. This limitation poses challenges in scenarios where infrastructure is scarce or compromised(such as post-disaster situations or remote mountainous areas) [4]. Therefore, there is a growing interest in leveraging Unmanned Aerial Vehicle (UAV) and Device-to-Device (D2D) assisted communications. These innovative approaches offer unique advantages in terms of mobility, adaptability, and coverage, which make them particularly suitable for addressing the limitations of traditional MEC deployments.

The research on UAVs primarily focuses on their trajectory and task scheduling [5]. UAV fly between fixed users to fulfill their computational offloading requirements. Meanwhile, there has been considerable research on the utilization of D2D communication as an emerging technology for MEC services [6]. The short-range and low-latency advantages of D2D communication assist MEC in task processing [7]. In the context of MEC servers, Virtual Machine (VM) reuse is a fundamental technique [8]. MEC servers utilize multiple VMs to perform parallel computing tasks, resulting in a significant reduction in computational latency. However, existing work has rarely explored the collaborative efforts of UAV-D2D communications and VM workloads on UAV to meet user task demands.

In this paper, we propose a system that leverages the collaborative advantages of UAV and D2D communication to assist MEC. UD splits and offloads tasks to UAV and D2D for joint calculation. We also optimize the workload on MEC servers to achieve optimal computational efficiency. The main contributions of this work are summarized as follows:

(1): We present the system of using UAV and D2D communication together to support MEC services. This novel approach harnesses the unique advantages of UAV mobility and D2D communication low latency. Compared to scenarios using only UAV or D2D, the task execution delay is reduced by 16% and 32%, respectively.

(2): We address the workload on MEC servers to maximize their computing capacity. By optimizing the allocation of VMs on the servers, we achieve a balanced workload and efficient resource utilization. Compared with no VMs allocation, task execution delay decrease 6%.

(3): We compare our proposed Deep Deterministic Policy Gradient (DDPG) with Deep Q-Network (DQN) and Actor-Critic (AC). Through simulations and experiments, we demonstrate that the DDPG reduces latency of tasks about 41%.

In the rest of this article is organized as follows. The Sect. 2 discusses related work. We introduced the system model in Sect. 3. In Sect. 4 we present the algorithm. The simulation and experimental results are presented in Sect. 5. The Sect. 6 summarizes.

2 Related Work

In recent years, there has been extensive research in the academic community focused on MEC assisted offloading. Arash *et al.* [9] aimed to minimize delay and energy consumption by finding the Pareto optimal frontier. Li *et al.* [10] proposed an online learning method that reduces task processing cost through multi-hop assisted collaboration. However, the static deployment of MEC servers mentioned above is not adaptable to various scenarios. Asim *et al.* [11] tackled the issue by minimizing system energy consumption through the optimization of the hovering position for each time slot of the Unmanned Aerial Vehicle (UAV). Wang *et al.* [12] employed Deep Reinforcement Learning (DRL) to plan multiple UAV trajectories while considering UAV load balancing to minimize energy consumption. Umber *et al.* [13] focused on D2D shared spectrum and aimed to minimize the sum of all device task execution delays under energy constraints, utilizing an offloading framework based on Orthogonal Frequency Division Multiple Access (OFDMA). Dai *et al.* [14] designed a framework that integrates migration and offloading willingness in D2D communication, aiming to minimize task latency and migration costs. However, there are limited studies on UAV and D2D co-assisted MEC computations. Pu *et al.* [15] found that opening multiple VMs in the same Physical Machine (PM) could impact overall performance due to I/O interference between VMs. Koushik *et al.* [16] employed the DQN algorithm, to design UAV trajectories and optimize network throughput. However, the DQN algorithm may face challenges in scenarios with continuous action spaces due to the curse of dimensionality, making convergence difficult. To address this, Ding *et al.* [17] proposed the DDPG algorithm to handle high-dimensional continuous motion of UAV and achieve improved performance.

In comparison to the reviewed related studies, we propose the DDPG algorithm to jointly leverage UAV and D2D communication for MEC offloading. Our approach aims to optimize the UAV trajectory and workload of MEC servers, leading to a reduction in task execution delay. By employing DDPG, we effectively address the challenge of high-dimensional continuous motion in UAV. In addition, offloading between close D2D is able to get low transmission latency. This jointly utilize advantages which enables efficient task distribution, ultimately resulting in a smaller task execution delay.

3 System Model

In this section, we will consider the issue of minimizing the latency of UDs. As shown in Fig. 1, we assume the D2D-assisted UAV-MEC system without BSs, which consists of a UAV and M UDs, denoted by the set $\mathcal{M} = \{1,2,\ldots,M\}$. Besides, We assume that the UDs are divided into two groups, one for D2D transmitter and one for D2D receivers, which set $\mathcal{I} = \{1, 2, \ldots, i, \ldots, I\}, \forall i \in \mathcal{M}$ and $\mathcal{J} = \{1, 2, \ldots, j, \ldots, J\}, \forall j \in \mathcal{M}$, respectively. The UAV is equipped with MEC servers and provides offloading services for the D2D transmitters. Simultaneously, the D2D receivers also assist in offloading tasks for the transmitters.

3.1 UAV Trajectory Model

In our model, we set a square region in Cartesian coordinates. Then, we assume that the UAV has sufficient power to maintain flying [2] at a fixed altitude H and serve the users dynamically during the flight cycle time of T. The flight period is divided into equal and sufficient small time slots N denoted by the set $\mathcal{N} = \{1,2,\ldots,n,\ldots,N\}$. Besides, We assume that the UAV remains hovering in each time slot n, so the position coordinate of the UAV is $\mathbf{q}(n) = [X(n), Y(n), H]$, $n \in \mathcal{N}$. The flight direction and speed is controlled by the angle of $\delta(n) \in (0, 2\pi]$ and $v(n) \in [0, v_{max}]$, respectively. Therefore, we get the coordinate of the UAV flies to the new hovering position at the nth time slot
$\mathbf{q}(n+1) = [X(n) + v(n)t\cos\delta(n), Y(n) + v(n)t\sin\delta(n), H]$
with a flight time t. Moreover, we have UAV flight constraints as the following

$$0 \le X(n+1) \le X(n) + v_{max}t\cos\delta(n), \forall n \in \mathcal{N} \tag{1}$$

$$0 \le Y(n+1) \le Y(n) + v_{max}t\sin\delta(n), \forall n \in \mathcal{N} \tag{2}$$

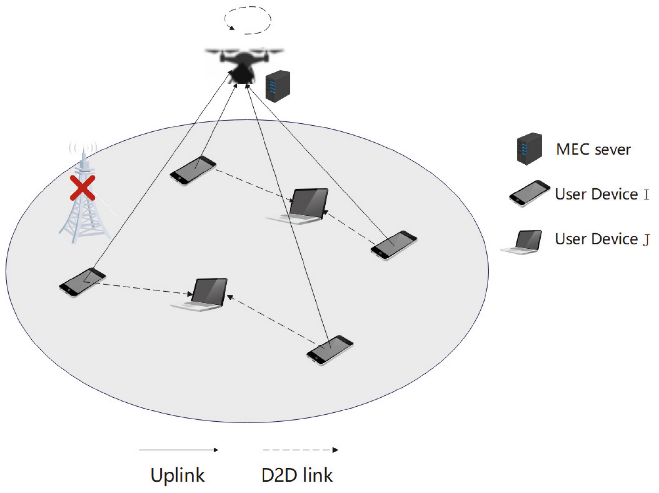


Fig. 1. UAV-D2D MEC System.

3.2 Communication Model

We assume that the UAV schedules one UD i per time slot to communicate, while this UD i generates a D2D link with the nearest UD j. Besides, we assume the positions of transmitter i and receiver j are fixed in our model, which are denoted as $\mathbf{q}_i = (x_i, y_i, 0)$ and $\mathbf{q}_j = (x_j, y_j, 0)$, respectively. The communication link between the UAV and the UD i is dominated by the line-of-site(LoS) channel, so their channel gain in time slot n be expressed as

$$g_i(n) = \beta_0 d_i^{-2}(n) = \frac{\beta_0}{\|\mathbf{q}(n) - \mathbf{q}_i\|^2}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I} \tag{3}$$

where β_0 is the channel power gain at a reference distance of 1 m, and $d_i(n)$ denotes the Euclidean distance between UD i and UAV. Moreover, we obtain the D2D channel link between UD i and UD j be modeled as

$$\begin{aligned} g_{i,j}(n) &= h(n)\beta_0 d_i^{-2}(n) \\ &= \frac{h(n)\beta_0}{\|\mathbf{q}_i - \mathbf{q}_j\|^2}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \end{aligned} \tag{4}$$

where $h(n)$ represents the small-scale fading coefficient of obeying $\mathcal{CN} \sim (0, 1)$. Then, the transmission rate between UAV and UD i is given as

$$r_i(n) = B_1 \log_2(1 + \frac{g_i(n)P_i(n)}{\sigma^2}), \forall n \in \mathcal{N}, \forall i \in \mathcal{I} \tag{5}$$

where B_1 is the ground-to-air channel bandwidth, $P_i(n)$ denotes the transmission power of the UD i, and σ^2 represents the noise power. Let B_2 denotes the ground-to-ground channel bandwidth, and the interference between links is ignored, so the data rate of the D2D link between UD i and UD j is given by

$$r_{i,j}(n) = B_2 \log_2(1 + \frac{g_{i,j}(n)P_{i,j}(n)}{\sigma^2}), \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{6}$$

3.3 Task Offloading Model

We assume that each time slot per UD i will generate different task $[D_i(n), V]$, where $D_i(n)$ denotes the task sizes, V indicates the CPU cycles to process each byte of the unit. Besides, we consider a partial offload mode, where a part of the tasks offload to the UAV and its ratio set to $R_i(n) \in [0, 1]$, and the remaining $(1 - R_i(n))$ is offloaded to the connected D2D device UD j. Because the size of the tasks returned after the calculation is very small, so they are usually negligible [11]. Therefore, the offloading transmission time from UD i to UAV at time slot n is

$$t_i^{tran}(n) = \frac{R_i(n)D_i(n)}{r_i(n)}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I}. \tag{7}$$

The transmission time from UD i to UD j is given as

$$t_{i,j}^{tran}(n) = \frac{(1 - R_i(n))D_i(n)}{r_{i,j}(n)}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \tag{8}$$

Hence, the transmission time of the scheduled UD i to offload tasks in time slot n is

$$T_{tran}(n) = t_i^{tran}(n) + t_{i,j}^{tran}(n), \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \tag{9}$$

3.4 Task Computing Model

In our model, we incorporate load balancing for the MEC server. As depicted in Fig. 2, upon receiving tasks, the MEC server creates multiple VMs on the same PM to process the tasks in parallel. However, turning on more VMs leads to increased load, which negatively impact the overall performance of the MEC server. We denote $Z > 0$ [18] as the attenuation factor, representing the percentage of overall computing capability degradation when multiple VMs are simultaneously active. Additionally, we assume that the tasks received by the MEC server can be randomly divided into multiple sub-tasks, with the number of sub-tasks denoted as $k(n)$ in time slot n. Consequently, the parallel computing time of $S(n)$ VMs on the MEC server in time slot n can be expressed as follows:

$$t_i^{comp}(n) = t_{\max}^{k(n)}(1+Z)^{S(n)-1}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I} \tag{10}$$

where $t_{\max}^{k(n)}$ denotes the maximum computing time for a VM to process parallel sub-tasks, and it is expressed as

$$t_{\max}^{k(n)} = \frac{D_{\max VM}^{sub}(n)V}{f_{VM}}, \forall n \in \mathcal{N} \tag{11}$$

where $D_{\max VM}^{sub}(n)$ represents the maximum sub-task size for a VM computing, f_{VM} denotes computing capability of virtual machine. Moreover, the computing time of UD j is

$$t_{i,j}^{comp}(n) = \frac{(1-R_i(n))D_i(n)V}{f_j}, \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \tag{12}$$

where f_j denoted the computing capability of UD j. Therefore, the total computing time of the task $D_i(n)$ at time slot n is

$$T_{comp}(n) = t_i^{comp}(n) + t_{i,j}^{comp}(n), \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}. \tag{13}$$

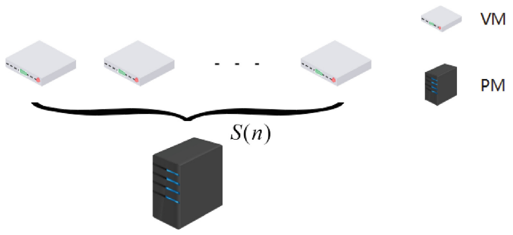


Fig. 2. VMs Parallel Computing.

3.5 Problem Formulation

In this paper, we jointly optimize user scheduling, UAV trajectory, UD i launch power, offload ratio, and number of VMs to achieve MEC server load balancing and minimize tasks execution latency. Specifically, we minimize the delay with transmission time and computation time which is formulated as

$$\min_{\substack{\alpha_i(n),\mathbf{q}(n+1),R_i(n),\\ P_i(n),P_{i,j}(n),S(n)}} \sum_{n=1}^{N}\sum_{i=1}^{I} \alpha_i(n)(T_{tran}(n)+T_{comp}(n)) \tag{14}$$

$$s.t. \quad \alpha_i(n) \in 0,1, \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \tag{14a}$$

$$\sum_{i=1}^{I} \alpha_i(n) = 1, \forall i \in \mathcal{I}, \tag{14b}$$

$$0 \le X(n+1) \le X(n) + v_{max} t \cos\delta(n), \forall n \in \mathcal{N}, \tag{14c}$$

$$0 \le Y(n+1) \le Y(n) + v_{max} t \sin\delta(n), \forall n \in \mathcal{N}, \tag{14d}$$

$$1 \le k(n) \le K_{max}, \forall n \in \mathcal{N}, \tag{14e}$$

$$1 \le S(n) \le k(n), \forall n \in \mathcal{N}. \tag{14f}$$

The constraint (14a) and (14b) ensure that only one user is scheduled for offloading in time slot n. Constraint (14c) and (14d) guarantee UAV flight trajectory is not exceeding its capacity limits. Constraint (14e) denotes the number of subtasks split on the MEC sever does not exceed the maximum. Constraint (14f) limit the number of VMs no more than the amount of sub-tasks.

4 Proposed Approach

In this section, so we propose a DRL algorithm DDPG to slove the above complex optimization problem with multiple non-convex constraints and multiple optimization objectives.

4.1 Algorithm DDPG

Reinforcement Learning (RL) methods involve an agent continuously interacting with the environment to determine the best action strategy for each step through trial and error [19]. When RL is combined with Deep Neural Networks (DNN), it forms DRL. Traditional DRL algorithms such as Q-learning, Sarsa, and DQN [20] are designed for problems with discrete action spaces. However, when dealing with continuous action spaces, the DDPG algorithm, as shown in Fig. 3, is utilized as a model-free off-policy AC [21] approach. In the DRL, the environment is typically modeled as a discrete-time Markov Decision Process (MDP). Following the Markov framework, the agent selects an action based on the current state of the environment and receives an immediate reward, which guides its subsequent actions. The primary objective of the agent is to maximize the accumulated reward by making optimal decisions based on the current environment state.

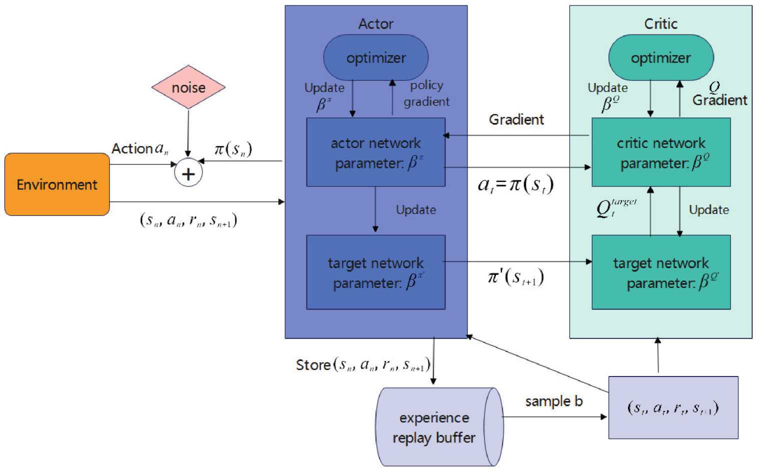

Actor
optimizer
Update β^{π}
policy gradient
actor network parameter: β^{π}
Update
target network parameter: $\beta^{\pi'}$
Critic
optimizer
Update β^{Q}
Q Gradient
critic network parameter: β^{Q}
Q_t^{target}
Update
target network parameter: $\beta^{Q'}$
Gradient
$a_t=\pi(s_t)$
$\pi'(s_{t+1})$
noise
Action a_n
$\pi(s_n)$
Environment
(s_n, a_n, r_n, s_{n+1})
Store (s_n, a_n, r_n, s_{n+1})
experience replay buffer
sample b
(s_t, a_t, r_t, s_{t+1})

Fig. 3. DDPG Schematic Diagram.

The DDPG as a deterministic policy algorithm where the continuous action spaces output is a deterministic action. The actor network π is defined as a function

$$a_n = \pi(s_n|\theta^{\pi}) \tag{15}$$

where s_n is current state to get deterministic action a_n, and θ^{π} is actor network training parameters. The critic network is $\mathcal{Q}(s_n, a_n|\theta^{\mathcal{Q}})$ to approximate Q-value function. Besides, both the actor network and the critic network contain the same structural target network as they are, which updates the approximate $\pi'(s_n|\theta^{\pi'})$ and $\mathcal{Q}'(s_n, a_n|\theta^{\mathcal{Q}'})$, respectively. In addition, DDPG has an experience replay mechanism that randomly selects the mini bitch b input network in the experience buffer to accelerate convergence. The critic network minimize the loss function to update

$$L(\theta^{\mathcal{Q}}) = \frac{1}{b}\sum_{t=1}^{b}[\mathcal{Q}^{target} - \mathcal{Q}(s_t, a_t|\theta^{\mathcal{Q}})]^2 \tag{16}$$

where $\mathcal{Q}^{target} = r_t + \gamma\mathcal{Q}(s_{t+1}, \pi'(s_t|\theta^{\pi'})|\theta^{\mathcal{Q}'}))$. Moreover, the strategy gradient update formula is

$$\nabla_{\theta^{\pi}} J = \mathbb{E}_{\pi'}[\nabla_a \mathcal{Q}(s, a|\theta^{\mathcal{Q}})|_{s=s_t, a=\pi(s_t|\theta^{\pi})} \nabla_{\theta^{\pi}} \pi(s|\theta^{\pi})|_{s=s_t}]. \tag{17}$$

Hence, the parameters of the target network are updated as

$$\theta^{\mathcal{Q}'} \leftarrow \tau\theta^{\mathcal{Q}} + (1-\tau)\theta^{\mathcal{Q}'} \tag{18}$$

$$\theta^{\pi'} \leftarrow \tau\theta^{\pi} + (1-\tau)\theta^{\pi'} \tag{19}$$

where $\tau \in (0, 1)$ is a constant to update target network softly.

Our DDPG is shown in Algorithm 1, we first initialize the network parametersthe (Algorithm 1: line 1–3). Then, we initialize the UAV state for each episode (Algorithm 1: line 4–5). UAV chooses action a_n in the actor network according to the state space. Because of the independence of exploration and learning in DDPG, we add the Gaussian noise N_n to the action exploration in order to avoid getting into local optimal solutions,

$$a_n = \pi(s_n|\theta^{\pi}) + N_n. \tag{20}$$

After executing action a_n based on s_n, the next state s_{n+1} and immediate reward r_n are observed. Then, the agent stores the transition four tuple in experience replay buffer (Algorithm 1: line 6–8). During training, if experience replay buffer B is full, the agent randomly selects b which sets of transition tuples in the buffer and puts them back into the network. The actor network and the critic network update their parameters to obtain the cumulative optimal reward (Algorithm 1: line 9–14). In the end, we get the best flight strategy for UAV (Algorithm 1: line 16–17).

Algorithm 1. DDPG-based Dynamic Computation Resource Allocation and Task Offloading algorithm

```
1: Initialize actor network with weights $\theta^{\pi}$ and critic network with weights $\theta^{Q}$.
2: Initialize the weights of target network $\theta^{\pi'} = \theta^{\pi}$ and $\theta^{Q'} = \theta^{Q}$, respectively.
3: Set the experience replay buffer $B = 0$.
4: for each episode do
5:     Reset the UAV initial position and observe the initial state $s_1$
6:     for $n = 1, 2, \ldots, N$ do
7:         Perform exploration actions $a_n = \pi(s_n|\theta^{\pi})+N_n$, get the reward $r_n$ and observe
           next state $s_{n+1}$.
8:         Store tuple $(s_n, a_n, s_{n+1}, r_n)$ in the experience replay buffer $B$.
9:         if $B$ is full, then
10:            Randomly sample tuple with mini-batches of $b$ from $B$.
11:            Update the $\theta^{Q}$ of critic network by minimizing the loss (16).Update the $\theta^{\pi}$
               of actor network by policy gradient (17).
12:            Update target network of critic network and actor network by (18) and
               (19), respectively.
13:        end if
14:    end for
15: end for
16: return $\theta^{\pi}$
17: Select the optimal action $a_n^{op}$.
```

4.2 MDP Model

In our system, UAV act as agent to creat MDP. We model the MDP as three tuples (S, A, R), where S indicates state space, A is a set of action, and R represents the reward function.

State Space. The state space of the environment in our model consists of UD i task size $D_i(n)$, the number of sub-tasks $k(n)$ and UAV position $\mathbf{q}(n)$ in time slot n. Therefore, the state space is given as

$$s_n = [D_i(n), k(n), \mathbf{q}(n)]. \tag{21}$$

Action Space. The action space consists of continuous flight actions and scheduling calculation of UAV, including flight speed $v(n)$, flight angle $\delta(n)$, offloading ratio $R_i(n)$, scheduling UD $i(n)$, UD launch power $P_i(n), P_{i,j}(n)$, and number of VMs $S(n)$. Thus, the action space is modeled as

$$a_n = [v(n), \delta(n), R_i(n), i(n), P_i(n), P_{i,j}(n), S(n)]. \tag{22}$$

Since the output actions of the actor network are continuous, the action variables $i(n)$, $S(n)$ need to be discretized, e.g. if $i(n) = 0$,then discretization $i' = 1$; if $i(n) \neq 0$, the $i' = \lceil i(n) \rceil$, where $\lceil \cdot \rceil$ is rounding up.

Reward Function. The reward function is a crucial component in evaluating the rationality of actions chosen by the agent. In our approach, we utilize the optimization objective as the basis for the reward function. Additionally, we incorporate a penalty factor, denoted as p_n, to account for the UAV flying out of the designated boundary. Consequently, the reward function can be expressed as follows:

$$r_n = -\sum_{n=1}^{N} \sum_{i=1}^{I} \alpha_i(n)(T_{tran}(n) + T_{comp}(n)) + p_n \tag{23}$$

Table 1. Simulation parameters

Parameter	Value	Parameter	Value
H	100 m	σ^2	–100 dBm
v_{max}	50 m/s	V	1000 cycles/bit
β_0	–50 dB	f_j	0.6 GHz
B_1	1 MHz	f_{VM}	1.2 GHz
B_2	0.8 MHz	Z	0.2
t	1 s	$D_i(n)$	[1.5, 2]Mbits

5 Simulation Results

In this section, we conduct simulations using the DDPG algorithm with specific parameter values, and compare its performance with other baseline algorithms. Our simulations use the CPU of AMD 5800H with 3.2 GHz. All algorithms are implemented in Python 3.6 and Tensorflow 1.5.0. The DDPG algorithm uses a 4-layer fully connected neural network with two hidden layers [300,10] neurons in both actor and critic networks. Our model considers a square area with dimensions of 100 m $\times$ 100 m. We have a total of $I = 4$ UDs positioned at [75, 19], [40, 88], [47, 17], and [93, 55] meters, respectively. Additionally, there are $J = 2$ destination points located at [0, 0] and [100, 100] meters. The initial position of the UAV is set to [50, 50] meters. For the simulation, we utilize various parameters which are specified as follows Table 1.

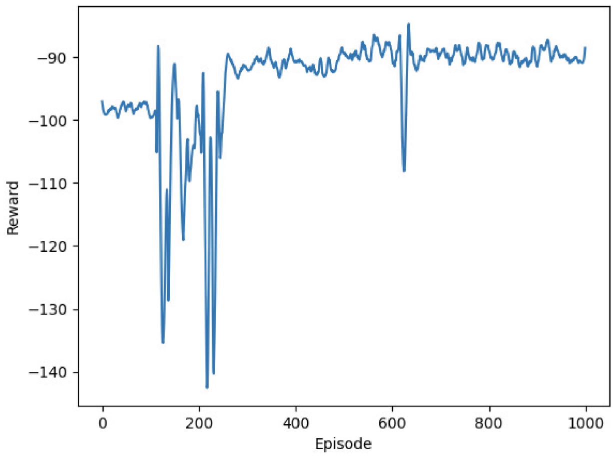


Fig. 4. Reward Convergence of DDPG.

First, we analyze the reward of DDPG as shown in Fig. 4. After conducting tests, we have observed that the best convergence performance when the learning rate of the critic network and the actor network is set to $\psi_{actor} = 0.001, \psi_{critic} = 0.002$, respectively. Meanwhile, our discount factor set as $\gamma = 0.001$, exploration parameter set as $\sigma_e = 0.01$. Initially, due to the lack of previous knowledge about the environment, the UAV explores actions in an almost random manner. As a result, the reward experiences significant fluctuations. However, as the UAV accumulates enough samples and gains more information about the environment, the reward gradually increases and eventually converges. This convergence indicates that the UAV has found the optimal flight strategy.

Furthermore, in Fig. 5, we compare the delay performance of various algorithms. As the training episodes increase, the AC algorithm fails to converge

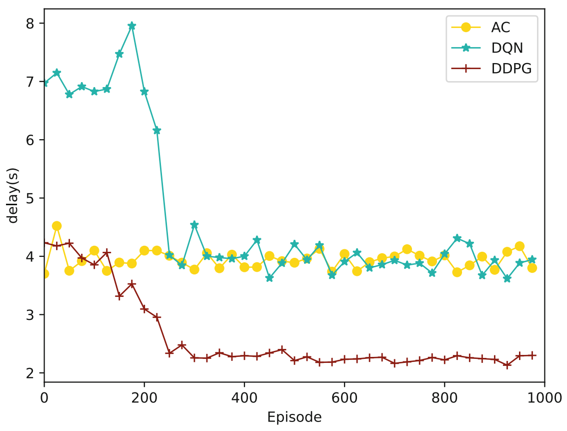
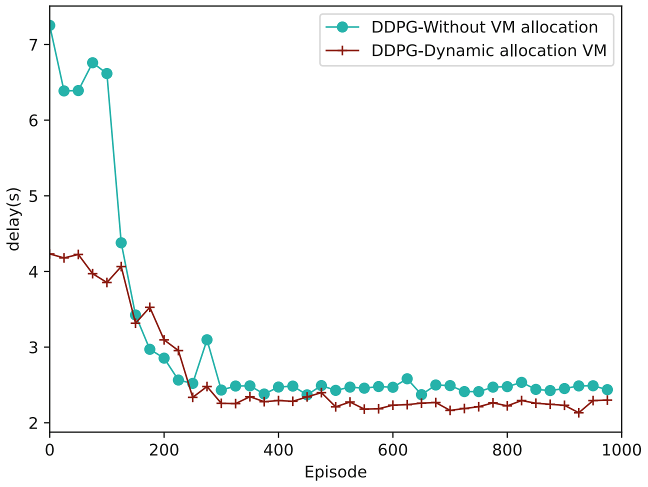


Fig. 5. Different Algorithm Delay Performance Comparison.

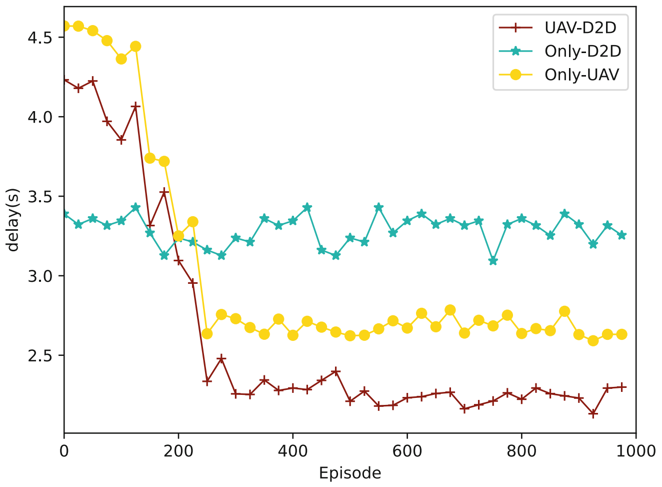
due to simultaneous updates of its actor and critic networks. The difficulty in converging the critic network prevents the accurate guidance of the optimal action through the value function. In contrast, both DQN and DDPG employ evaluation networks and target networks, which ensure relatively independent training data and enable convergence. Because the DQN is limited to scenarios with discrete actions, when dealing with problems involving a large number of action dimensions, we need to quantify continuous actions into finite discrete values which will result in lower cumulative rewards. The DDPG converges to smaller delay values eventually because of the extensive exploration of continuous actions. As a result, DDPG has 41% lower latency than DQN and AC.

Figure 6 (a) illustrates a comparison of the delay performance between DDPG with and without dynamic VM allocation. We observe that both systems converge after 300 episodes as the UAV finds the best strategy. However, the allocation of VM results in lower task processing latency. This is because optimizing the distribution of VM leads to a further reduction in computational latency, causing a decrease in latency of 6%.

In Fig. 6 (b), we simulate the delay performance of UAV or D2D working alone, and compare them with our system. When only D2D is working, due to the limited computing capability of D2D receiver devices, they are unable to quickly complete all task processing. Besides, only relying on UAV operation unable guarantee low-latency transmission for all tasks. Our model allows the user to partially offload to the UAV and partially offload to the D2D receiver device, which leverages the respective strengths of UAV and D2D. Therefore, the latency of our model is 16% or 32% smaller than when only UAV or D2D works alone, respectively.



(a) Delay Performance of DDPG with and without VM Allocation.



(b) Comparison of UAV-D2D combined and solo operation delay.

Fig. 6. Delay comparison of different system models.

6 Conclusion

In this paper, we consider a system that D2D and UAV-MEC collaborate to assist in user task offloading. We optimize the UAV trajectory and achieve load balancing on the MEC server to minimize the sum of the tasks' transmission delay and computation delay. Specifically, to solve the integer nonlinear problem, we propose the DDPG algorithm to obtain the optimal strategy.

Through extensive simulations, we evaluate the performance of DDPG in terms of processing task latency and compare it with DQN and AC. The results demonstrate that DDPG approach outperforms the DQN and AC about 41% in terms of task latency reduction. This indicates the effectiveness and superiority of our proposed solution in optimizing the UAV trajectory and achieving load balancing on the MEC server.

By leveraging the collaboration between D2D and UAV-MEC, our system demonstrates improved efficiency and reduced latency in task offloading. And compared with only UAV or D2D wokrs alone, the task processing delay reduces 16% and 32%, respectively. In addition, the latency for dynamic VM allocation is 6% lower than for fixed VM numbers.

Acknowledgment. This work was supported in part by the Ningbo Natural Science Foundation under Grant 2021J070, in part by the Zhejiang Natural Science Foundation under Grant LY20F010004, and National Natural Science Foundation of China under Grant 61801254.

References

1. Cui, G., He, Q., Chen, F., Zhang, Y., Jin, H., Yang, Y.: Interference-aware Game-Theoretic device allocation for mobile edge computing. IEEE Trans. Mob. Comput. **21**(11), 4001–4012 (2022)
2. Zhang, J., Wang, Z.-J., Wang, K., Guo, S., Wang, B., Guo, M.: Improving power efficiency for online video streaming service: a self-adaptive approach. IEEE Trans. Sustain. Comput. **4**(3), 308–313 (2019)
3. Tiankui Zhang, Y.X., Loo, J., Yang, D., Xiao, L.: Joint computation and communication design for UAV-assisted mobile edge computing in IoT. IEEE Trans. Ind. Inf. **16**(8), 5505–5516 (2020)
4. Li, X., Feng, W., Chen, Y., Wang, C., Ge, N.: UAV-Enabled accompanying coverage for hybrid satellite-UAV-terrestrial maritime communications. In: Proceedings of 28th Wireless and Optical Communications Conference, pp. 1–5 (2019)
5. Wang, Z., Duan, L., Zhang, R.: Adaptive deployment for UAV-aided communication networks. IEEE Trans. Wirel. Commun. **18**(9), 4531–4543 (2019)
6. Chatzopoulos, D., Bermejo, C., Haq, E.U., Li, Y., Hui, P.: D2D task offloading: A dataset-based Q and A. IEEE Commun. Mag. **57**(2), 102–107 (2019)
7. Cheng, Y., Liang, C., Chen, Q., Yu, R.: Energy-efficient D2D-assisted computation offloading in NOMA-Enabled cognitive networks. IEEE Trans. Veh. Technol. **70**(12), 13441–13446 (2021)
8. Liang, Z., Liu, Y., Lok, T., Huang, K.: Multiuser computation offloading and downloading for edge computing with virtualization. IEEE Trans. Wireless Commun. **18**(9), 4298–4311 (2019)
9. Bozorgchenani, A., Mashhadi, F., Tarchi, D., Salinas Monroy, S.A.: Multi-objective computation sharing in energy and delay constrained mobile edge computing environments. IEEE Trans. Mob. Comput. **20**(10), 2992–3005 (2021)
10. Li, Y., Wang, X., Gan, X., Jin, H., Fu, L., Wang, X.: Learning-Aided computation offloading for trusted collaborative mobile edge computing. IEEE Trans. Mob. Comput. **19**(12), 2833–2849 (2020)

11. Asim, M., Mashwani, W.K., Shah, H., Belhaouari, S.B.: A Load-Balanced and Energy-Efficient navigation scheme for UAV-Mounted mobile edge computing. Soft Computing, pp. 1–14. Springer, Berlin, Germany (2021)
12. Wang, Z., Rong, H., Jiang, H., Xiao, Z., Zeng, F.: An evolutionary trajectory planning algorithm for multi-UAV-assisted MEC system. IEEE Trans. Netw. Sci. Eng. **9**(5), 3659–3674 (2022)
13. Saleem, U., Liu, Y., Jangsher, S., Tao, X., Li, Y.: Latency minimization for D2D-Enabled partial computation offloading in mobile edge computing. IEEE Trans. Veh. Technol. **69**(4), 4472–4486 (2020)
14. Dai, X., et al.: Task Co-Offloading for D2D-Assisted mobile edge computing in industrial Internet of Things. IEEE Trans. Ind. Inf. **19**(1), 480–490 (2023)
15. Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., Pu, C.: Understanding performance interference of I/O workload in virtualized cloud environments. In: Proc. IEEE 3rd International Conference on Cloud Computing, pp. 51–58 (2010)
16. Koushik, A.M., Hu, F., Kumar, S.: Deep Q-learning-based node positioning for throughput-optimal communications in dynamic UAV swarm network. IEEE Trans. Cogn. Commun. Netw. **5**(3), 554–566 (2019)
17. Ding, R., Gao, F., Shen, X.S.: 3D UAV trajectory design and frequency band allocation for energy efficient and fair communication: a deep reinforcement learning approach. IEEE Trans. Wireless Commun. **19**(12), 7796–7809 (2020)
18. Liu, Y., Yan, J., Zhao, X.: Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime UAV communication network. IEEE Trans. Veh. Technol. **71**(4), 4225–4236 (2022)
19. Orhean, A.I., Pop, F., Raicu, I.: New scheduling approach using reinforcement learning for heterogeneous distributed systems. J Parallel Distrib Comput. **117**, 292–302 (2018)
20. Mnih, V., Kavukcuoglu, K., Silver, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
21. Cheng, N., et al.: Space/Aerial-assisted computing offloading for IoT applications: a learning-based approach. IEEE J. Sel. Areas Commun. **37**(5), 1117–1129 (2019)