Future Generation Computer Systems 180 (2026) 108389

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Online 3D trajectory and resource optimization for dynamic UAV-assisted MEC systems

Zhao Tong[a], Shiyan Zhang[a], Jing Mei[a,*], Can Wang[a], Keqin Li[b,c,d]

[a] College of Information Science and Engineering, Hunan Normal University, Changsha, 410081, China
[b] College of Information Science and Engineering, Hunan University, Changsha, 410082, China
[c] National Supercomputing Center in Changsha, Changsha, 410082, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Mobile edge computing
ODEGD
Resource allocation
Trajectory planning
Unmanned aerial vehicle
User mobility.

ABSTRACT

The integration and development of unmanned aerial vehicles (UAVs) and mobile edge computing (MEC) technology provide users with more flexible, reliable, and high-quality computing services. However, most UAV-assisted MEC model designs mainly focus on static environments, which do not apply to the practical scenarios considered in this work. In this paper, we consider a UAV-assisted MEC platform, which can provide continuous services for multiple mobile ground users with random movements and task arrivals. Moreover, we investigate the long-term system utility maximization problem in UAV-assisted MEC systems, considering continuous task offloading, users' mobility, UAV's 3D trajectory control, and resource allocation. To address the challenges of limited system information, high-dimensional continuous actions, and state space approximation, we propose an Online decision-making algorithm for Dynamic environments based on Exploration-enhanced Greedy DDPG (ODEGD). Additionally, to more accurately evaluate the algorithm's performance, we introduced real-world roads into the experiment. Experimental results show that the proposed algorithm reduces response delay by 26.98% and energy consumption by 22.61% compared to other algorithms, while achieving the highest system utility. These results validate the applicability of the ODEGD algorithm under dynamic conditions, demonstrating its good robustness and scalability.

1. Introduction

With the rapid development of Internet-of-Things (IoT) technology, the number of mobile devices continues to surge, driving the rapid growth of new types of mobile applications (i.e., automatic navigation, AR/VR virtual technology, online interactive games, etc.) [1,2]. These applications are often computationally intensive and sensitive to response time. Due to limited computing power and battery capacity, user equipment (UE) cannot provide high-performance computing for these tasks. Mobile edge computing (MEC) [3] offers a cost-effective solution, that transmits computing requests to the edge of MEC systems equipped with servers (i.e., cellular base stations or WiFi access points). Edge servers provide computing resources to meet the computationally intensive tasks of UE, reducing task response time, extending device battery life, and improving user experience quality.

Nevertheless, there may be scenarios where a large number of UEs require computing services concurrently, in which case MEC system resources are insufficient to provide services, especially in hotspot areas [4]. Additionally, in areas with inadequate communication infrastructure or limited coverage like suburbs, it's unknown whether UE can access the MEC system to receive services [5]. Consequently, ensuring the provision of reliable offloading services remains a challenging issue. Unmanned aerial vehicles (UAVs), due to their flexibility and maneuverability, have been regarded as playing a critical role in enhancing future wireless communication systems. UAVs equipped with small servers can serve as aerial computing platforms, providing flexible and elastic computing services for UEs, with the advantages of efficient and convenient deployment, wide coverage, and high cost-effectiveness. For instance, [6,7] applies a UAV platform to improve MEC coverage and energy efficiency of a large number of IoT devices in disaster areas or remote regions. Another scenario is in the field of wireless sensors, where drones are used to collect data from sensor nodes or offload computing tasks on sensors [8,9]. Therefore, the UAV-assisted MEC technology has become one of the key technologies of the 5G network.

This paper integrates mobile UEs, the 3D flight-capable UAV, and DRL into a dynamic MEC environment by optimizing the response delay

* Corresponding author.
E-mail addresses: tongzhao@hunnu.edu.cn (Z. Tong), 202370294159@hunnu.edu.cn (S. Zhang), jingmei1988@163.com (J. Mei), wangcan@hunnu.edu.cn (C. Wang), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.future.2026.108389
Received 22 June 2025; Received in revised form 7 January 2026; Accepted 16 January 2026
Available online 24 January 2026

and energy consumption of task offloading. By utilizing the computing resources of edge servers, the computing pressure on UEs can be alleviated within the constraints of ensuring continuous service quality. The bi-objective optimization of Online decision-making in Dynamic environments based on Exploring enhancing Greedy DDPG (ODEGD) algorithm is designed. The major contributions of our work are summarized as follows:

- We formulate an online offloading decision model that accounts for both the mobility of UEs and their continuous offloading needs. This model jointly optimizes the UAV's 3D trajectory and computational resource allocation to maximize system utility. It is difficult to find the optimal strategy in a dynamic environment where the future state of the system is unclear.
- We propose an effective algorithmic framework based on the partially observable Markov decision process (POMDP), a robust mathematical model for sequential decision-making. By using deep neural networks, we standardize and approximate the limited agent's observations to a complete state space, and we integrate the DDPG algorithm to handle the continuous action space. In this highly complex optimization problem with a large solution space, we also introduce greedy strategies to enhance the framework's exploration capability.
- We introduce real-world roads into the experiments to evaluate the algorithm's practical performance. The numerical results demonstrate that our framework outperforms the baseline optimization algorithms, particularly in terms of UE perception capability, robustness to environmental changes, and adaptability to simulation durations and UE count growth. Furthermore, the combination of the UAV's 3D mobility and our framework further enhances the overall system utility.

The remainder of this paper is organized as follows: Section 2 is devoted to related work. Section 3 presents the system model and the problem formulation. In Section 4, the ODEGD algorithm is introduced. Section 5 analyzes the performance of the proposed algorithm through experimental results. Lastly, Section 6 concludes this paper.

2. Related work

Current research on UAV-assisted MEC systems focuses on computation offloading [10–12], resource allocation [13–15], and trajectory planning [16–18]. Masuduzzaman et al. proposed a UAV-based MEC-assisted automated traffic management framework that integrates blockchain for secure data sharing and deep learning for vehicle detection, achieving decentralized and intelligent traffic control [19]. Zhang et al. developed a power cognition framework for solar-powered UAVs that leverages reinforcement learning to optimize energy harvesting, trajectory control, and resource allocation for long-endurance and high-throughput communications [20]. Most studies consider the situation where the position or flight altitude of the UAV is fixed [21,22]. Due to the close relationship between communication quality and the movement trajectory of UAVs, different movement patterns of UAVs may affect the comprehensive distance with UEs, resulting in different communication delays and energy consumption. Moreover, some studies adopt a binary offloading approach, which may result in the waste of total system computing resources [23,24]. How to consider the needs of UEs and the status of system resources, and reasonably divide and transmit computing tasks to UAVs, which will have a significant impact on computing latency and energy consumption. In addition, almost all existing research focuses on offline algorithm design for UAV-assisted MEC systems in static environments [25,26]. These algorithms assume that the UEs' location is fixed, UEs' computation needs remain unchanged, or are known in advance, and optimize the offloading strategy and movement trajectory based on this assumption. However, in many MEC scenarios, the UE will move and the computing tasks will change. This situation requires the design of an online decision-making algorithm for UAV-assisted MEC systems, which can make real-time decisions when facing

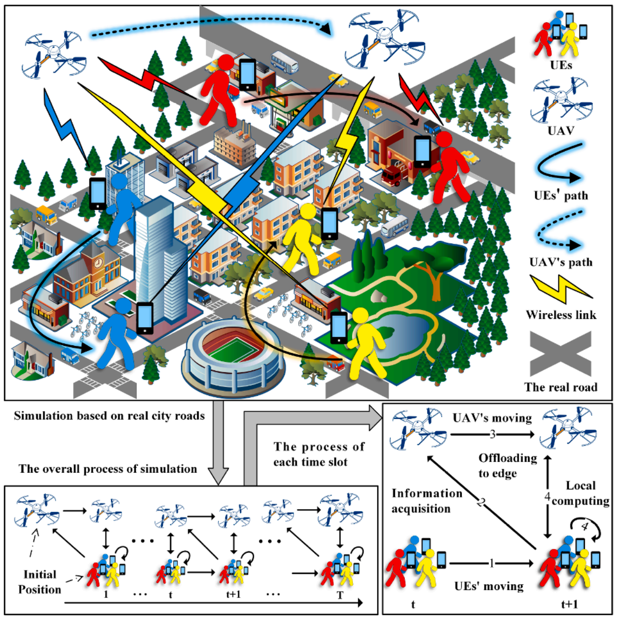


Fig. 1. UAV-assisted MEC system model.

unknown future system states. Therefore, it is necessary to consider the trajectory of UAVs, the allocation of computing tasks, and dynamic environmental changes together to achieve the smallest response delay and energy consumption.

In recent years, researchers have proposed many algorithms to solve different optimization problems in UAV-assisted MEC systems. Chai et al. formulated the path planning of the UAV in the sensor nodes (SNs) data collection problem as the traveling salesman problem (TSP) and solved it using the simulated annealing (SA) algorithm [26]. Yang et al. designed an online algorithm based on Lyapunov optimization to solve queue stability problems in UAV-assisted MEC systems [27]. Nguyen et al. proposed using deep reinforcement learning (DRL) to optimize data collection in UAV-assisted MEC systems, achieving significant overall rate improvement and maximum resource utilization with UAV's limited battery capacity [28]. Zhao et al. integrated the extended kalman filter (EKF) with the proximal policy optimization (PPO) algorithm to jointly optimize vehicle trajectory prediction and task offloading decisions in Internet of Vehicles systems, effectively reducing delay and energy consumption [29]. Hu et al. proposed a framework based on dynamic programming (DP) and ant colony (AC) heuristic algorithms to minimize the average age of information (AoI) [30]. Wang et al. combined the successive-convex-approximation (SCA) with block coordinate descent (BCD) technology to solve the problems of data collection and energy consumption in the system [31]. These methods have achieved certain results in solving non-convex optimization problems. However, as the problem becomes more complex and the solution space expands, these methods may be difficult to solve. Meanwhile, facing non-convex problems and non-stationary environments, it can be challenging to achieve a globally optimal strategy without precise and comprehensive knowledge of the environment.

3. System model and problem formulation

In this section, we first propose an MEC model considering dynamic environments. Then, the models of mobile, computing, and communication were introduced in the following subsections. Finally, the problems that need to be addressed within this model are formulated.

As shown in Fig. 1, this model consists of two layers of MEC architecture in a 3D environment, including M mobile UEs and a UAV equip a small server. The UAV contains communication and computing services, providing UEs with continuous offloading services. The total simulation time in this model is N, which is partitioned into discrete time slots, that is, $N = T \times \Delta t$, where T denotes the number of time slots and Δt denotes the intervals of time slot. The UE generates a small task $W_m(t)$ consisting of the size of the task $D_m(t)$ and the number of CPU cycles $C_m(t)$ in each time slot t, i.e. $W_m(t) = \left[D_m(t), C_m(t)\right]$. These tasks can be computed locally or partially or completely offloaded to the UAV. Therefore, the UAV needs to make online offloading decisions to enhance service quality. Moreover, the UAV moves around the UEs to make sure all UEs in the signal coverage can be serviced by the UAV.

3.1. *Mobility model*

3.1.1. *UE Mobility model*

In this UAV-assisted MEC scenario, the position model is modeled as a stereoscopic Cartesian coordinate system and the coordinates of UEs are updated in each time slot. $c_m(t) = \left[x_m(t), y_m(t), 0\right]$ was used to represent the coordinates of UE m in time slot t. At next time slot $t+1$, UE m will move to new location with coordinates $c_m(t+1)$, and the movement distance of UE m is represented as $\Delta d_m(t) = \left\|c_m(t+1) - c_m(t)\right\|$. UEs move once in each time slot, following the selected road. After completing its path, they choose a new road to move on, simulating the movement pattern on real-world roads.

Assuming that at the start of each time slot t, the UAV obtains each UE's position coordinates $\left\{c_m(t)\right\}_{m=1}^{M}$ through communication. However, the UAV doesn't know the future trajectory coordinates and task parameters of the UE, which represented as $\left\{c_m(i), W_m(i)\right\}_{m=1}^{M}, \forall i \in \{t+1, \dots, T\}$. Therefore, the environment of the entire model is dynamic. To optimize resource utilization, the UAV needs to sense the motion pattern of UEs and predict the location where UEs may need computing resources in the future.

3.1.2. *UAV Mobility model*

The coordinates of the UAV at time slot t are represented as $c_u(t) = \left[x_u(t), y_u(t), z_u(t)\right]$, where $x_u(t)$ and $y_u(t)$ represent the UAV's horizontal position, and $z_u(t)$ denotes the UAV's flight altitude.

Assuming that in time slot t, the UAV moves a distance of $\Delta d_u(t)$ in the horizontal direction at an angle of $\omega(t) \in [0, 2\pi)$ and flies the distance $\Delta z_u(t)$ in vertical direction. Its 3D coordinates at time slot $t+1$ are updated as follows:

$$x_u(t) = x_u(t-1) + \Delta d_u(t)\cos(\omega(t)), \tag{1}$$

$$y_u(t) = y_u(t-1) + \Delta d_u(t)\sin(\omega(t)), \tag{2}$$

$$z_u(t) = z_u(t-1) + \Delta z_u(t). \tag{3}$$

This 3D mobility enables the UAV to efficiently adjust its position by sensing the location of UEs, providing high-quality computing and communication services for each UE. In addition, the flight process is much shorter than the service process, allowing the UAV to promptly perform subsequent position adjustments. This real-time adaptability enables the UAV to effectively handle various UEs' tasks. Denote $v_u(t) = \left[x_u(t), y_u(t)\right]$ as the UAV's 2D coordinates. Then, the movement distance of the UAV in each time slot t is limited by

$$\Delta d_u(t) = \left\|v_u(t) - v_u(t-1)\right\| \le \Delta D_{\max}, \tag{4}$$

$$\left|\Delta z_u(t)\right| = \left|z_u(t) - z_u(t-1)\right| \le \Delta Z_{\max}, \tag{5}$$

where $\Delta D_{\max}$ and $\Delta Z_{\max}$ respectively represent the maximum movement distance of the UAV in the horizontal and vertical directions.

Furthermore, we have designated a rectangular service area to ensure that the signal range of the UAV can cover all UEs. The movement of the UAV cannot exceed the boundaries of the service area, that is,

$$0 \le x_u(t) \le X_{\max}, \tag{6}$$

$$0 \le y_u(t) \le Y_{\max}, \tag{7}$$

where $X_{\max}$ and $Y_{\max}$ respectively represent the length and width of the rectangle-shaped service area.

Similarly, the flight altitude of the UAV must be restricted. This is to prevent the limited coverage capability of the UAV from being adversely affected by flying too high, which could lead to poor service quality. It also avoids the situation in which the UAV may collide with buildings when flying too low. The UAV's flight altitude must not exceed the specified minimum heights $Z_{\min}$ and maximum heights $Z_{\max}$. Then, we have the following constraint

$$Z_{\min} \le z_u(t) \le Z_{\max}. \tag{8}$$

3.2. *Task computation model*

This model adopts partial offloading, which means that a portion or the entirety of a task will be offloaded and processed on the UAV, using the $\xi_m^u(t)$ to represent the offloading ratio. If a task requires computation, the portion $\xi_m^u(t)$ is executed by a computing node on the UAV, while the remaining $\left(1 - \xi_m^u(t)\right)$ part is processed locally on the UE. When tasks are computed locally or on the edge, the corresponding cost will be generated, including response delay and energy consumption. In addition, a task can only be offloaded to one computing node on the UAV, indicating that during the offloading process, a task can't be simultaneously assigned to multiple computing nodes. In each time slot, the UAV will intelligently select suitable offloading strategies based on the location information of all UEs, optimizing task allocation and utilization of computing resources. Therefore, by continuously monitoring the UEs' location information in real-time, the UAV can provide more efficient continuous computing services.

3.2.1. *Local computing at UE*

The UE contains some computing power and is capable of handling some of the tasks. The computational delay consumed by the UE to process the task in the time slot t is expressible as

$$T_{local}^m(t) = \frac{\left(1 - \xi_m^u(t)\right)C_m(t)D_m(t)}{F_m}, \tag{9}$$

where F_m represents the UE's computational resources.

Then, according to the above computational delay, the energy expended by UE m during the above calculation process can be given as

$$\begin{aligned} E_{local}^m(t) &= \kappa F_m^3 T_{local}^m(t) \\ &= \kappa F_m^2\left(1 - \xi_m^u(t)\right)C_m(t)D_m(t), \end{aligned} \tag{10}$$

where κ denotes the effective capacitor switch.

3.2.2. *Edge computing at UAV*

After receiving task data from the UEs, the UAV begins to perform calculations. Since the UEs in our scenario are assumed to be of the same type, the tasks they generate in different time slots have similar characteristics, such as task size and the number of required CPU cycles, which fluctuate within a relatively stable range. Therefore, to simplify the model, we assumed that the computational resources of UAV's server F_u are evenly distributed to all UE [17,26,32]. This assumption facilitates analytical tractability without significantly affecting the evaluation of the proposed algorithm's effectiveness. At the current time slot, the computational delay for processing UE m's task is expressible as

$$T_{uav}^m(t) = \frac{\xi_m^u(t)C_m(t)D_m(t)}{F_u^m}, \tag{11}$$

where F_u^m represents the computational resources assigned by the UAV to UE m, that is, $F_u^m = F_u / M$.

Similarly, the energy expended by the server configured on the UAV during the above calculation process can given as:

$$\begin{aligned} E_{uav}^m(t) &= \kappa\left(F_u^m\right)^3 T_{uav}^m(t) \\ &= \kappa\left(F_u^m\right)^2 \xi_m^u(t)C_m(t)D_m(t). \end{aligned} \tag{12}$$

3.3. Task transmission model

Offloading tasks to the edge involves not only computational costs but also communication costs. When part of the task is selected to transfer to the UAV for computation, the communication between UAV and UEs uses the orthogonal frequency division multiplexing access. Therefore, communication interference between each UE is ignored in this model.

In this model, the UAV remains within a certain high altitude range, so we assumed that all channels belong to the line-of-sight channel, i.e., there is no occlusion between UEs and the UAV. Therefore, we use the free-space path loss model to obtain the channel gain between UE m and UAV in time slot t, that is,

$$h_m^u(t) = g_0 \left[l_m^u(t)\right]^{-2}, \tag{13}$$

where g_0 denotes the channel gain at a distance of 1m, and $l_m^u(t)$ represents the current Euclidean distance between UE m and UAV:

$$l_m^u(t) = \left\|c_u(t) - c_m(t)\right\|. \tag{14}$$

The scenario of this model involves UAV-assisted task offloading in a real urban environment, which includes a lot of noise, both natural and artificial. Therefore, the transmission loss caused by these noises on the uplink has been considered. Additionally, like the UAV's computational resources F_u, we assumed that the bandwidth in the upload link B_u is evenly allocated among all UEs. Therefore, the uplink transmission rate of the task data is denoted as

$$R_m^u(t) = \frac{B_u}{M}\log_2\left[1 + \frac{P_m h_m^u(t)}{\sigma^2 + \delta}\right], \tag{15}$$

where P_m represents the UE's transmission power of the uplink, σ^2 represents the noise power, and δ represents the transmit loss [33].

The transmission delay between the UAV and UE m during time slot t can be expressed as the time spent on uploading the selected task to the UAV, that is,

$$T_{trans}^m(t) = \frac{\xi_m^u(t) D_m(t)}{R_m^u(t)}. \tag{16}$$

Similarly, the energy consumed by the transmission process between the UAV and UE m is given as

$$E_{trans}^m(t) = \left(P_m + P_u\right) T_{trans}^m(t), \tag{17}$$

where P_u denotes the UAV's receiving power.

In the UAV-assisted MEC system, the size of task results returned sent back by the UAV is minimal, thus the impact of downlink transmission delay and energy consumption can be neglected.

3.4. Problem formulation

To achieve the optimal offloading strategy and UAV's trajectory planning, an optimization model is built that takes energy consumption and response delay as the objective function. In each time slot t, the response delay for one UE's task depends on the longer task delay in the local and edge computations, and the energy consumption is the sum of the computations and transmissions. When all UEs' tasks in time slot t are completed, the total response delay for that time slot, denoted as $T(t)$, can be expressed as

$$T(t) = \sum_{m=1}^{M} \max\left\{T_{local}^m(t), T_{trans}^m(t) + T_{uav}^m(t)\right\}, \tag{18}$$

and the total energy consumption, denoted as $E(t)$, is given as

$$E(t) = \sum_{m=1}^{M} \left[E_{local}^m(t) + E_{trans}^m(t) + E_{uav}^m(t)\right]. \tag{19}$$

To facilitate a more effective comparison of algorithms' optimization levels, we introduce baselines for response delay and energy consumption separately. $T_{baseline}$ and $E_{baseline}$ represent the response delay and energy consumption when the model only performs local computation without offloading, i.e.,

$$T_{baseline} = \sum_{t=1}^{T}\sum_{m=1}^{M} \frac{C_m(t) D_m(t)}{F_m}, \tag{20}$$

$$E_{baseline} = \sum_{t=1}^{T}\sum_{m=1}^{M} \kappa F_m^2 C_m(t) D_m(t), \tag{21}$$

With study [34,35] similarly, the response delay reduction value obtained by consuming energy/unit is taken as the system utility. Therefore, the system utility in this model is denoted as

$$Utility = \frac{T_{baseline} - \sum_{t=1}^{T} T(t)}{\sum_{t=1}^{T} E(t) - E_{baseline}}. \tag{22}$$

Consequently, the objective of this model is to achieve maximum system utility by optimizing the UAV's location $c_u(t)$ and task offloading ratios $\xi_m^u(t)$ in each time slot. The formulated optimization problem is denoted as follows:

$$\max_{c_u(t), \xi_m^u(t)} \; Utility, \tag{23}$$

$$s.t. \;\; 0 \le \xi_m^u(t) \le 1, \forall m, \tag{23a}$$

$$(4) - (8), \tag{23b}$$

where constraint (23a) denotes the range of task allocation ratio, (23b) includes the maximum mobility constraint and service area boundary constraint of the UAV.

Generally, the key challenge of non-convex optimization problems is the possibility of multiple local optimal solutions, complicating the search for the global optimal solution. In the above practical scenario, the UAV itself needs to make online decisions on 3D movement and offloading service, while the dynamic environment for the UEs' mobility and the tasks' randomness. These challenges result in the optimization problem (23) having a large solution space, making it difficult to solve through traditional optimization solutions. To tackle these challenges, a DRL method that obtains the optimal offloading strategy with less information in a dynamic environment was proposed in this paper.

4. ODEGD for task offloading optimization problem

In this section, the partially observable Markov decision process is used to rephrase the bi-objective optimization problem, and its essential elements are defined. Then, the ODEGD framework was introduced and applied to solve optimization problems.

4.1. POMDP formulation

The UAV's trajectory planning and UEs' task offloading are essentially a decision-making problem, which can be formulated as a Markov decision process (MDP) problem and then solved using reinforcement learning methods. This model refers to a real urban scenario, where the UAV needs to make real-time decisions in a dynamic environment. That means the UAV cannot get the position information of UEs at all time slots. Therefore, the bi-objective optimization problem can be represented as a partially observable Markov decision process $\langle S, A, T, R, \Omega, O, \gamma\rangle$. Here, S denotes the state space of the environment, A denotes the action space of the agent, T represents the probability of state transition, R denotes the reward function, Ω denotes the observation space of the agent, O represents the probability of observe transition, and $\gamma \in [0, 1]$ represents the discount factor.

4.1.1. State space

In time slot t, the system environment contains various information such as the UAV's 3D coordinates $c_u(t)$, UEs' position $c_m(t)$, and UEs' task information $W_m(t)$. Therefore, the state space S of the system environment is denoted as

$$s(t) = \left\{c_u(t), c_m(t), W_m(t), \forall m\right\}. \tag{24}$$

We can observe that coordinate elements have three dimensions, while task information has two dimensions. When multiplied by the number of UEs M, this leads to a high degree of dimensionality in the state space. Moreover, the elements within the state $s(t)$ do not share the same value range.

4.1.2. *Observation space*

In time slot t, the agent's observation of the complete environment state space $s(t)$ is expressed as

$$\Omega(t) = \left\{c_u(t), c_m(t), W_m(t), \forall m\right\}. \tag{25}$$

When the environment is in state $s \in S$ and action $a \in A$ is taken, the agent considers the system environment to have changed to $\tilde{s}'$. However, since the UEs' movement is unknown, the $\tilde{s}'$ obtained by the agent does not match the real environmental state, i.e., $\tilde{s}' \neq s', s' \in S$. To address this issue, the partially observable Markov decision process uses the observation o' to represent $\tilde{s}'$, that is, $o' = \tilde{s}', o \in \Omega$. Therefore, the state space $s(t)$ and the observation space $\Omega(t)$ contain the same elements but different transition probability functions. The probability of the agent taking action a causing the environment to transition from state s to state s' is represented as $T(s,a,s') = P\left(s'|s,a\right)$. Meanwhile, the observation o of the agent depends on the new state s' of the environment, and the probability is expressed as $O\left(s',a,o\right) = P\left(o\middle|a,s'\right)$.

4.1.3. *Action space*

After observing the current state $s(t)$, the agent is required to make decisions including horizontal movement distance $\Delta d_u(t)$, horizontal movement angle $\omega(t)$, vertical movement distance $\Delta z_u(t)$, and offloading rate $\xi_m^u(t)$ for each UE. The action space A at time slot t can be represented as

$$a(t) = \left\{\Delta d_u(t), \Delta z_u(t), \omega(t), \xi_m^u(t), \forall m\right\}. \tag{26}$$

The range of values for each element in action $a(t)$ can be obtained based on the maximum optimization problem (23), i.e, $\Delta d_u(t) \in \left[0, \Delta D_{\max}\right]$, $\Delta z_u(t) \in \left[-\Delta Z_{\max}, \Delta Z_{\max}\right]$, $\omega(t) \in [0, 2\pi)$, and $\xi_m^u(t) \in [0,1]$. Meanwhile, we can observe that as the number of UEs increases, the dimensions of the state space and action space rapidly increase.

4.1.4. *Reward function*

When action $a(t)$ is performed in state $s(t)$, the agent receives a reward from the environment, which represents the quality of action $a(t)$. Therefore, the reward function $R(t)$ guides the agent in learning the appropriate policy. This model aims to reduce the cost consisting of response delay and energy consumption of tasks in each time slot t, which can be represented as $Cost(t) = \omega_1 T(t) + \omega_2 E(t)$, to maximize the overall system utility. Similar to [32,36], we use ω_1 and ω_2 to unify the dimensions of the two variables. Therefore, the reward function $R(t)$ is defined as the negative value of the system's cost at time slot t, that is,

$$R(t) = -Cost(t). \tag{27}$$

4.2. *ODEGD framework*

In practice, the partially observable Markov decision process problems are generally difficult to solve computationally. Some researchers define a probability distribution in the state space called Belief State, which infers the probability of currently being in state s based on historical information. Then simplify the problem into a similar Markov decision process problem through the belief state and solve it. However, this method requires a certain or even a large amount of storage resources to record historical information. Therefore, we use deep neural networks to obtain an approximate space $\hat{S}$ that approximates the state space S by observing space Ω. Then the above issue can be simplified to an Markov decision process problem and addressed using solutions based on reinforcement learning. During the training process of neural networks, significant disparities in the units of measurement and value ranges of training samples may lead to vanishing or exploding gradients, which can adversely affect the learning efficiency of the model. In this model, the variables $c_u(t), c_1(t), c_2(t), \ldots, c_M(t), W_1(t), W_2(t), \ldots, W_M(t)$ in the observe space Ω lie in different ranges, which may make the training process more difficult. To mitigate this issue, this study employs a normalization algorithm as shown in Algorithm 1. This technique not only enhances the model's convergence performance but also bolsters the model's adaptability and robustness. The model incorporated five scaling factors, corresponding to the range spanned by the maximum and minimum values for each variable. Each factor can be explained as follows. Given that the UAV and UEs share the same range for their x and y coordinates, we employ ξ_X and ξ_Y to normalize their x and y positions. The scaling factor ξ_Z normalizes the UAV's z coordinate. And ξ_D and ξ_C are used to normalize UEs' task size and CPU cycles respectively.

Algorithm 1: The Normalization Algorithm.

```
Input: Un-normalized observe variables: $c_u(t)$, $c_1(t)$, $c_2(t)$, …, $c_M(t)$, $W_1(t)$, $W_2(t)$, …, $W_M(t)$. Scaling factor: $\xi_X$, $\xi_Y$, $\xi_Z$, $\xi_D$, $\xi_C$.
Output: Normalized observe variables: $c'_u(t)$, $c'_1(t)$, $c'_2(t)$, …, $c'_M(t)$, $W'_1(t)$, $W'_2(t)$, …, $W'_M(t)$.
1 $x_u{}'(t) = x_u(t)/\xi_X$;
2 $y_u{}'(t) = y_u(t)/\xi_Y$;
3 $z_u{}'(t) = z_u(t)/\xi_Z$;
4 for $m = 1$ to $M$ do
5     $x_m{}'(t) = x_m(t)/\xi_X$, $y_m{}'(t) = y_m(t)/\xi_Y$;
6     $D_m{}'(t) = D_m(t)/\xi_D$, $C_m{}'(t) = C_m(t)/\xi_C$;
7 end
8 return $c'_u(t)$, $c'_1(t)$ …, $c'_M(t)$, $W'_1(t)$, …, $W'_M(t)$.
```

After transforming the partially observable Markov decision process problem through neural networks, reinforcement learning algorithms can be used to solve it. Q-Learning is a fundamental and classic algorithm in reinforcement learning, with the core idea of constructing and maintaining a Q-table. When solving continuous or even complex discrete problems, Q- Learning consumes a significant amount of resources to store all Q-values, making it impractical. Combining deep learning with reinforcement learning, the deep Q-network (DQN) algorithm introduces neural networks to replace the Q-table, thus solving the issue mentioned above.

Nevertheless, DQN suffers from overestimation of Q-values and unstable training, making it unsuitable for problems with continuous action spaces. Deep deterministic policy gradient (DDPG) [37] is an off-policy DRL algorithm that is built upon the Actor-Critic framework. It integrates Value-based learning with Policy-based learning by drawing on the ideas of DQN and deterministic policy gradient. At the same time, it introduces target networks to ensure stable training of the networks. Therefore, DDPG consists of four networks: the actor network $\pi(s|\theta^{\mu})$, the target actor network $\pi'(s\middle|\theta^{\mu'})$, the critic network $Q(s,a\middle|\theta^{Q})$, and the target critic network $Q'(s,a\middle|\theta^{Q'})$. Therefore, we integrate the DDPG algorithm into the ODEGD framework to solve the online offloading problem in a dynamic environment. A two-stage method is used to approximate the state space S using neural networks, followed by solving the problem using the DQN algorithm [38]. This method requires training two sets of neural networks and adjusting their respective hyperparameters. In contrast, we let the ODEGD algorithm undertake these two tasks simultaneously, that is, using the same set of neural networks for state space approximation and optimal solution search. Therefore, we only need to train a set of neural networks and adjust their hyperparameters.

Algorithm 2: ODEGD-Based Task Offloading Algorithm.

```
Input: Trajectory Dataset of UEs.
Output: System total utility.
1  Initialize the actor network $\pi([o,\hat{s}]|\theta^{\mu})$ and critic network $Q([o,\hat{s}],a|\theta^{Q})$;
2  Initialize the each target networks $\pi'([o,\hat{s}]|\theta^{\mu'})$, $Q'([o,\hat{s}],a|\theta^{Q})$: $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$;
3  Initialize the replay buffer $B$;
4  for $each\ episode$ do
5      Set $t=1$ and initialize observe $o(t)$;
6      for $t=1$ $to$ $T$ do
7          Normalize $o(t)$ to $o'(t)$ according to alg. 1;
8          if $rand(\cdot) < \varepsilon$ then
9              Randomly select action $a(t)$;
10         else
11             $a(t) = clip(\pi([o'(t),\hat{s}(t)]|\theta^{\mu}) + \tilde{n}, -1, 1)$;
12         end
13         Obtained reward $r(t)$ and next observe $o(t+1)$;
14         Normalize $o(t+1)$ to $o'(t+1)$;
15         Store $\langle o'(t), a(t), r(t), o'(t+1)\rangle$ in $B$;
16         Sample a random mini-batch $B_s$ from $B$;
17         Calculate the target Q-value and TD-error using (29) and (30);
18         Update critic network parameters $\theta^{\mu}$ using (31);
19         Update actor network parameters $\theta^{Q}$ using(32);
20         Soft update target network parameters $\theta^{\mu'}, \theta^{Q'}$ using (33) and (34);
21     end
22 end
23 return
```

In addition, balancing exploration and exploitation is an important issue in reinforcement learning. DDPG algorithm lets the agent explore by adding noise to actions, i.e. $a = \pi(s'|\theta^{\mu}) + \tilde{n}$, where $\tilde{n} \sim N(0, \hat{\sigma}^2)$ is the action noise with $mean = 0$ and standard deviation $\hat{\sigma}$. However, the model in this paper not only requires the agent to search for the optimal strategy in the multidimensional action space but also demands the approximation of the state space. This situation greatly increases the difficulty of exploring the solution space for optimization problems, and noise methods may not be able to find the optimal solution. Therefore, we introduce the $\varepsilon - greedy$ strategy into the ODEGD algorithm, allowing the agent to choose random actions with a ε probability, or take actions given by the network with a $1-\varepsilon$ probability. While ODEGD is built upon the standard DDPG framework, it incorporates an exploration-enhancing greedy strategy and an observation-state approximation process to improve decision stability, robustness, and convergence performance in dynamic UAV-assisted MEC environments.

Algorithm 2 summarized the ODEGD framework for addressing the bi-objective optimization problem in a dynamic environment. First, four networks' parameters and the replay buffer B are initialized. Each episode begins with the reset of observations and time slots. Then, in each time slot, we normalize the observe of the UAV. Based on the $\varepsilon - greedy$ strategy, the UAV selects a random action or action composed of the output given by the actor network $\pi([o'(t), \hat{s}(t)]|\theta^{\mu})$ plus Gaussian noise $\tilde{n}$. As delineated in the action space definition, the UAV will move horizontally in distance $\Delta d_u(t)$ at an angle $\omega(t)$, move vertically in distance $\Delta z_u(t)$, and partially offload each UE's task in proportion $\xi_m^u(t)$. If the UAV's movement exceeds the prescribed flight range, which does not meet Constraint (6) to (8), it will keep flying at the boundary. Before storing the transition in the replay pool, we also normalize the observations of the UAV at $t+1$ to ensure stable training. To ensure the stability of the training process, we also normalized the UAV's next observations $o'(t+1)$. Then we save the transition tuple $\langle o'(t), a(t), r(t), o'(t+1)\rangle$ in the replay buffer B for improve sample learning efficiency.

The ODEGD algorithm requires optimizing Actor networks to maximize the value of the policy evaluation and Critic networks to minimize the error of the value evaluation. The role of the actor network $\pi([o,\hat{s}]|\theta^{\mu})$ is to provide a policy that can determine what action a should be taken in the observe o. The better the actor network's policy, the better the actions it can provide. The actor network is optimized through policy gradient, as the formula below:

$$\nabla_{\theta^{\mu}} J(\theta^{\mu}) \approx \frac{1}{B_s}\sum_{i=1}^{B_s} \nabla_a Q([o,\hat{s}],a|\theta^{Q})|_{o_i, a=\pi(o_i)} \nabla_{\theta^{\mu}} \pi([o,\hat{s}]|\theta^{\mu})|_{o_i}, \tag{28}$$

where B_s denotes the batch size. From the formula, it can be seen that the policy gradient depends on the critic network $Q([o,\hat{s}],a|\theta^{Q})$, whose role is to score the action a given by the actor network in observe o. The more accurate the critic network scoring, the more correct the direction of optimization for the actor network's update.

The critic network is optimized through the value function. Using the target actor network $\pi'([o,\hat{s}]|\theta^{\mu'})$ and the target critic network $Q'([o,\hat{s}],a|\theta^{Q'})$, the target Q-value Q_{target} can be given by

$$Q_{\text{target}} = r(t) + \gamma(Q'([o(t+1),\hat{s}(t+1)], \pi'([o(t+1),\hat{s}(t+1)]|\theta^{\mu'})|\theta^{Q'}), \tag{29}$$

where γ is the discount factor, indicating the importance of future rewards. Then, the temporal difference error (TD-error) of the critic network is expressed as

$$L(\theta^{Q}) = \frac{1}{B_s}\sum_{i=1}^{B_s}\left[Q_{\text{target}} - Q\left([o,\hat{s}],a|\theta^{Q}\right)\right]^2. \tag{30}$$

Finally, gradient ascent and gradient descent methods are used to optimize the actor and critic networks, respectively, i.e.,

$$\theta^{\mu} \leftarrow \theta^{\mu} + \alpha_{actor}\nabla_{\theta^{\mu}} J(\theta^{\mu}), \tag{31}$$

$$\theta^{Q} \leftarrow \theta^{Q} - \alpha_{critic}\nabla_{\theta^{Q}} L(\theta^{Q}), \tag{32}$$

where α_{actor}, α_{critic} represent the learning rate, which controls the weight between the old and new network parameters.

In addition, we use the soft update method to update all target networks, ensuring the stability of the algorithms, denoted as

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \tag{33}$$

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{34}$$

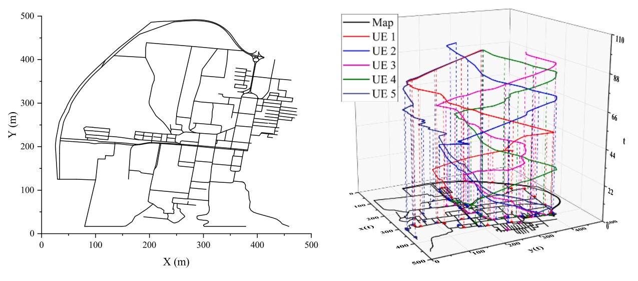
where τ denotes the soft update rate, $\tau \in [0,1]$. This soft update step, commonly used in machine learning, is similar to an under-relaxation process in classical numerical optimization. Introducing the $\varepsilon - greedy$ strategy to enhance exploration may lead to a long-term trial process, resulting in low training efficiency. Therefore, we have adopted a single step update approach for all networks, while setting a reasonable greedy initial value that decreases with training.

5. Experiment results and analysis

In this section, the UEs' trajectory dataset based on the map of real-world roads and settings of the experiment is first introduced. Then, we select an appropriate set of hyperparameters for conducting the subsequent experiments. Finally, the ODEGD algorithm was compared with DRL baseline algorithms.

5.1. Trajectory dataset of the UEs

In the existing UAV-assisted MEC systems, UE's data for simulation experiments is usually generated using theoretical assumptions or manually generated methods. These methods can, to a certain extent, test



(a) Map of real-world roads (b) UEs' trajectory

Fig. 2. The map used by the ONE simulator and movement trajectory of different UEs.

Table 1
Experimental simulation parameters.

Parameter	Value
Total simulation time N	60 s
Time interval Δt	3 s
Number of UE M	5
Size of input data D_m	[1,5] Mbits [100,200]
Number of CPU cycles C_m	cycles/bit
Lengths of area $X_{\max}$,$Y_{\max}$	500 m
Flight altitude of UAV [$Z_{\min}$,$Z_{\max}$]	[50,100] m
Movement distance of UE $\Delta d_m(t)$	[13,17] m
Maximum horizontal distance $\Delta D_{\max}$	49 m
Maximum vertical distance $\Delta Z_{\max}$	12 m
Channel gain g_0	-50 dB
Uplink channel bandwidth B_u	10 MHz
Transmit and receiving power P_m,P_u	0.1 W
Computation resource of UEs F_m	0.2 GHz
Computation resource of UAV F_u	3 GHz
Effective switched capacitance κ	10^{-28}
Noise power σ^2	-100 dBm
Transmit loss δ	20 dB
Location scaling factor ξ_X, ξ_Y, ξ_Z	500, 500, 50
Task scaling factor ξ_D, ξ_C	4×10^6, 100

Table 2
Main hyperparameters.

Hyperparameters	Value
Total episodes	1000
Learning rate α_{actor}, α_{critic}	0.001, 0.002
Discount factor γ	0.001
Size of replay buffer B	100,000
Batch size B_s	128
Standard deviation of noise $\hat{\sigma}$	0.05
Soft update rate τ	0.005
Activation function	ReLU
Loss function	Mean-square error
Optimizer	AdamOptimizer

the performance of algorithms. Nevertheless, there is always a discrepancy between generated data and actual data, which may lead to bias in algorithm performance evaluation. To more accurately reproduce the mobility patterns of UEs and overcome the sim-to-real gap, this paper adopts a trajectory and task of the UEs dataset based on real-world roads. As shown in Fig. 2(a). The opportunistic network environment (ONE) simulator [39] is an open-source software developed by the University of Helsinki in Finland, which is extensively utilized for generating UEs' movement trajectories employing diverse mobility models. The real map was imported into the ONE simulator in this paper to generate a dataset of motion trajectories for UEs, including a training set of 1 trajectory file and a testing set of 2500 trajectory files. Fig. 2(b) presents the movement trajectories of five UEs over 100 time slots, highlighting their mobility patterns.

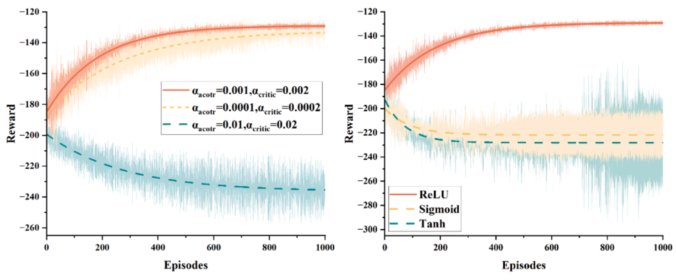
5.2. Experimental settings

In the UAV-assisted MEC system, the simulated scene is an area of $X \times Y \times Z = 500 \times 500 \times 100$ m^3 square with $M = 5$ mobile UEs. The UAV used for task offloading can only move between altitude $Z_{\min} = 50$ m and altitude $Z_{\max} = 100$ m. The size of input data D_m and the number of CPU cycles C_m are randomly sampled within [1, 5] Mbits and [100, 200] cycles/bit, respectively. Similar to [40], transmit loss due to the buildings is set to $\delta = 20$ dB. The scaling factors used in Algorithm 1 are defined as $\xi_X = 500$, $\xi_Y = 500$, $\xi_Z = 50$, $\xi_D = 4 \times 10^6$, $\xi_C = 100$. In both the training and testing phases of the algorithm, the UAV's starting position is randomly determined, unless specifically noted otherwise. The detailed parameters of our system are listed in the Table 1.

Apart from the introduced ODEGD approach, we also utilized five additional optimization methods to compare performance:

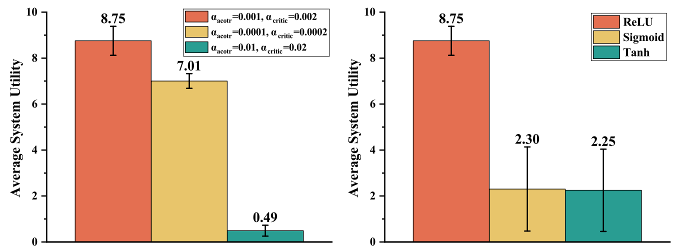
1) Deep Q-Network (DQN): A classical reinforcement learning algorithm employs deep neural networks to approximate the Q-value function and is commonly used for discrete action space problems. In this algorithm, the movement and offloading of the UAV are discretized. The horizontal fly angle is expressed as $\omega(t) = \{0, \pi/4, \ldots, 7\pi/4\}$, the horizontal and vertical movement distance are set as $\Delta d_u(t) = \{0\text{m}, 3\text{m}, \ldots, 12\text{m}\}$ and $\Delta z_u(t) = \{0\text{m}, 12\text{m}, 24\text{m}, 36\text{m}, 49\text{m}\}$, and the offloading ratio can be defined as $\xi^u_m(t) = \{0, 1/2, 1\}$.
2) Double Deep Q-Network (DDQN): An improved algorithm designed to address the issue of overestimation that may occur in DQN. It has two structurally identical networks: Evaluate-network for selecting actions and Target-network for calculating the target Q-value. This improvement can reduce network overestimation and provide a more stable learning process. The discretization of action space in this algorithm is consistent with DQN.
3) Actor-Critic Network (AC): AC combines the policy gradient method and value function, consisting of two networks: Actor-network and Critic-network. The Actor network selects actions based on the current policy, while the Critic network is responsible for evaluating the actions chosen by the actor-network. Compared with DQN and DDQN algorithms, the AC algorithm can solve continuous action space problems and improve sample utilization efficiency.
4) Twin Delayed Deep Deterministic Policy Gradient (TD3): TD3 is a further improvement based on DDPG, using two sets of Critic networks. The smaller of the two is taken when calculating the target Q value. Both action choice and strategy gradients incorporate noise to enhance exploration. In addition, Critic networks update more frequently than Actor networks to maintain stability. However, such improvements increase the complexity of the structure and do not suit all problem environments.
5) Ant Colony Optimization (ACO): ACO is a heuristic optimization algorithm inspired by the foraging behavior of ants, where agents cooperatively construct solutions through pheromone-guided searches. It has shown strong performance in solving discrete optimization problems by balancing exploration and exploitation through heuristic updates. However, ACO is difficult to address continuous action space problems; therefore, its action space discretization setting is consistent with DQN.

The above five algorithms have differences in network structure, usage scenarios, and function types. To ensure a fair comparison with ODEGD, all algorithms were implemented using the same network architecture and normalization scheme, and each was carefully tuned to its optimal hyperparameters. Note that the baseline DDPG algorithm employs the standard Ornstein-Uhlenbeck (OU) noise for exploration. Therefore, the performance comparison between ODEGD and DDPG also reflects the contribution of the proposed exploration mechanism, which enhances robustness and stability in dynamic UAV-assisted MEC environments. In addition, the simulation experiment was conducted on a computer with a 3.80GHz AMD Ryzen 7 5800X 8-Core Processor and



(a) Learning rate (b) Activation function

Fig. 3. Training curves of the ODEGD-based algorithm with different hyperparameters.



(a) Learning rate (b) Activation function

Fig. 4. Average system utility of the ODEGD-based algorithm with different hyperparameters.

32GB RAM, based on Python 3.10 and TensorFlow framework. The average of 5 experimental results is taken as the final result for the simulation experiment.

5.3. *Training efficiency*

In the ODEGD-based algorithm, hyperparameters including the learning rate α, discount factor γ, batch size B_s, and the choice of activation function play crucial roles in determining the efficiency of the training process. Therefore, before comparing the performance of different algorithms, we first conduct several experiments to determine the hyperparameters required by the algorithms.

The training curves of the proposed algorithm across various learning rates are depicted in Fig. 3(a). In the ODEGD algorithm, the learning rates applied to the actor and critic networks vary, yet they are of the same order of magnitude. Firstly, we can see no sign of convergence in $\alpha_{\text{actor}} = 0.01$, $\alpha_{\text{critic}} = 0.02$ because a higher learning rate will result in a larger update step for the network, leading to skipping the optimal solution. Secondly, when using a lower learning rate like $\alpha_{\text{actor}} = 0.0001$, $\alpha_{\text{critic}} = 0.0002$, the convergence speed of the algorithm significantly slows down. When the learning rate is set to $\alpha_{\text{actor}} = 0.001$, $\alpha_{\text{critic}} = 0.002$, it can be seen that the algorithm converges around 600 episodes. From another perspective, the system utility of $\alpha_{\text{actor}} = 0.001$, $\alpha_{\text{critic}} = 0.002$ is significantly higher than other learning rates, as shown in Fig. 4(a), where the error bars (as well as those in the subsequent experimental figures) represent the standard deviations of the experimental data. The mean utilities of the three configurations were 8.75 ± 0.59, 7.01 ± 0.32, and 0.49 ± 0.24, respectively. A two-sample *t*-test indicated that $\alpha_{\text{actor}} = 0.001$, $\alpha_{\text{critic}} = 0.002$ significantly outperformed $\alpha_{\text{actor}} = 0.0001$, $\alpha_{\text{critic}} = 0.0002$ (t(8) = 5.55, p < 0.001, 95% confidence interval (CI) [1.02, 2.46], Cohen's d = 3.63). Therefore, we adopt $\alpha_{\text{actor}} = 0.001$, $\alpha_{\text{critic}} = 0.002$ as the optimal learning rate for the actor and critic network.

Activation functions are a key component of neural networks that introduce nonlinear factors to the network to capture complex patterns and relationships. Some commonly used activation functions are ReLU, Sigmoid, and Tanh. As shown in Figs. 3(b), 4(b), we compare the training efficiency and average utility with the abovementioned activation functions. It can be observed that only the training curve of ReLU converges, while Sigmoid and Tanh diverge more in the later stages of training. That is because ReLU is simple and efficient, which can relieve the problem of gradient vanishing. Sigmoid and Tanh involve exponential operations, so when the absolute value of the input is large, the gradient approaches 0, which can easily lead to the gradient vanishing during backpropagation. As can be seen from the system utility, ReLU's 8.75 ± 0.59 is significantly better than Sigmoid's 2.30 ± 1.83 and Tanh's 2.25 ± 1.79. Therefore, we chose ReLU as the algorithm's activation function.

In addition, we also compared experiments with different batch sizes and discount factors. The experimental results indicate that these two hyperparameters have almost no difference in convergence effect and system utility, so we did not display these two sets of graphs. Finally, the main experimental hyperparameters are compiled in Table 2.

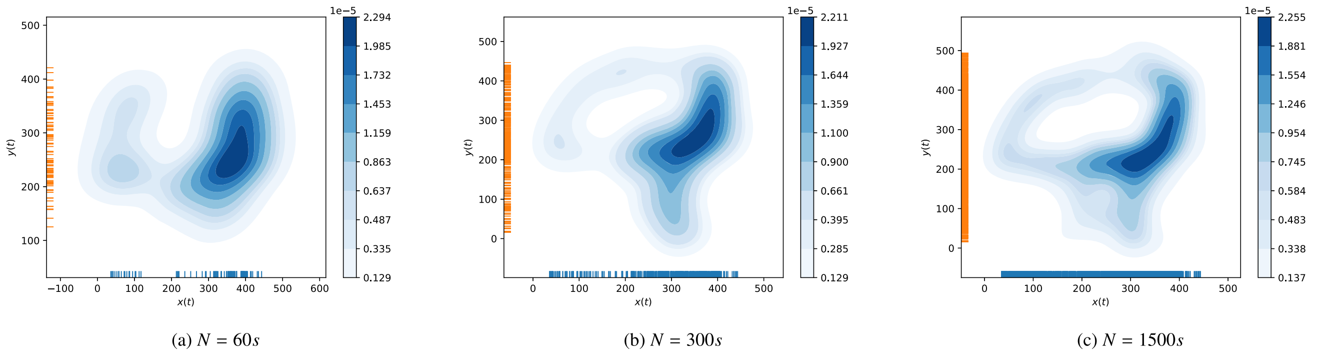
Fig. 5 shows the frequency of occurrence of UEs' positions in the training set at three specific time points (60 seconds, 300 seconds, and 1500 seconds), indicating significant clustering in UE location distribution. At 60 seconds, the UEs' movement trajectory is concentrated at x = 350m, y = 250m to 350m; At 300 seconds, we can see that the UE's trajectory appears in a relatively remote area, but the frequency is low. At 1500 seconds, the distribution characteristics of UEs' positions are more obvious, such as point (350,250) and its surrounding areas are hot spots for UEs. These UEs' trajectory thermal maps provide the basis for subsequent analysis and inspection of the UAV movement path.

After training the agent through the ODEGD method, for the training set file, the agent can perform optimal movement and offloading operations in each time slot t. Starting from point (250, 250, 75), the hotspots of the UAV's location at each specific time point are shown in Fig. 6. We can observe that the hotspot area of the UAV trajectory is concentrated near the point (270,240), which almost overlaps with the high-frequency gathering centers of UEs, indicating that our algorithm can perceive the gathering areas of UEs through their movement trajectories. However, it is also worth noting that the UAV hotspots exhibit a slight leftward bias relative to the UE's centers, about 70 m, suggesting that the agent's service strategy has also been optimized for UEs on more remote roads. From the y-axis direction, the flight trajectory area of the UAV gradually expands, because the UAV will move according to the distribution of the Y-axis of the UEs in different time slots, rather than staying in place.

We have set four different departures to simulate the flexible deployment of UAV: start point A (500, 500, 100), start point B (500, 0, 100), start point C (0, 500, 100), and start point D (0, 0, 100). Fig. 7 presents the UAV's trajectory from these various starting points. It can be observed that after departing from deployment points, the algorithm plans the UAV's path based on the UAV's current position and the obtained UEs' orientation and movement pattern, ultimately reaching the optimal service area. Furthermore, by examining the commonalities and characteristics of the four trajectories, we can see that the UAV can quickly descend in altitude during the initial simulation period to reduce communication distance with UEs, while also adjusting behavior details appropriately according to its location and environment. These demonstrate that the ODEGD algorithm has good robustness in random environments.

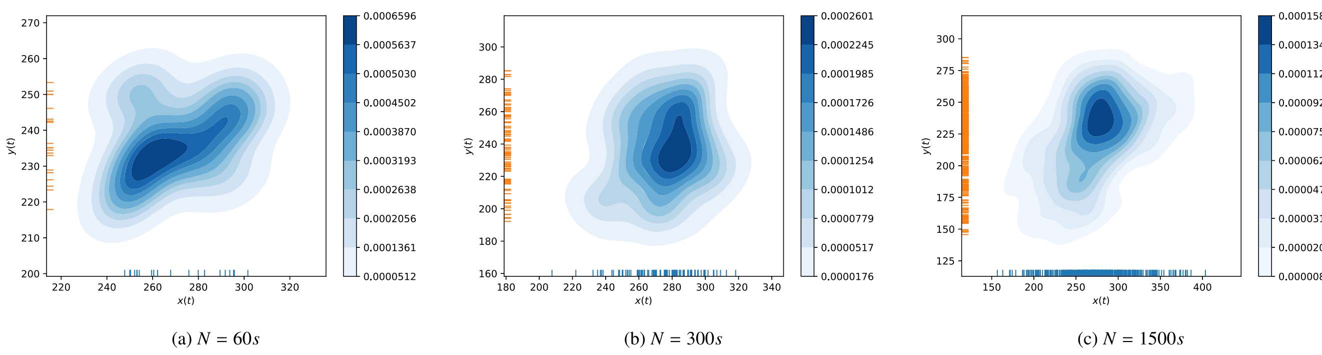
5.4. *Performance comparison*

To verify the advantages of our proposed ODEGD algorithm, we compared its performance with four DRL baseline algorithms. Fig. 8(a) plots the training curves of different DRL algorithms. From this figure, we can observe that DQN and DDQN fail to converge as the number of episodes increases. This is likely because these two algorithms are not well-suited for optimization problems with complex action spaces. The ACO algorithm does not exhibit an optimization trend. Hence, in the following comparisons, the episode achieving the highest reward is regarded as



(a) $N = 60s$ (b) $N = 300s$ (c) $N = 1500s$

Fig. 5. UEs' position frequency heatmap with different simulation times.



(a) $N = 60s$ (b) $N = 300s$ (c) $N = 1500s$

Fig. 6. UAV's 2D position frequency heatmap with different simulation times.

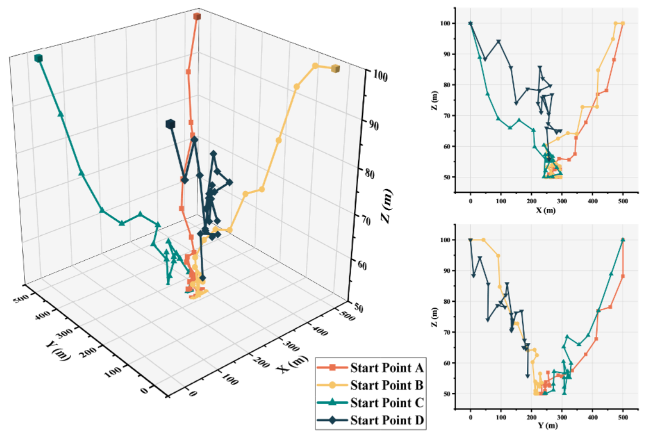
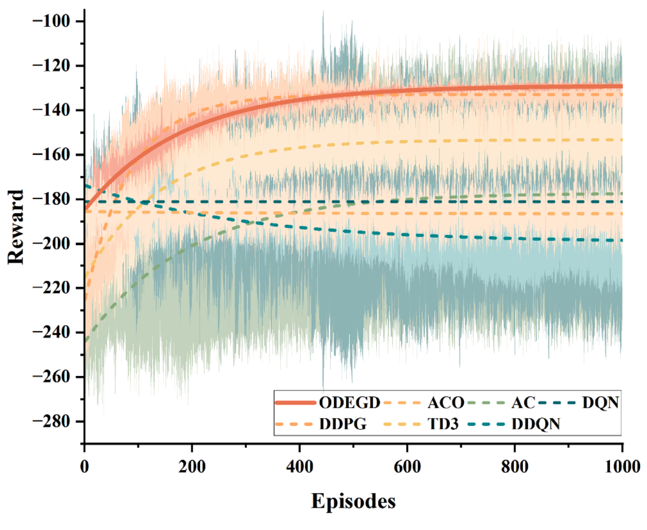


Fig. 7. UAV's 3D trajectory from different start point.

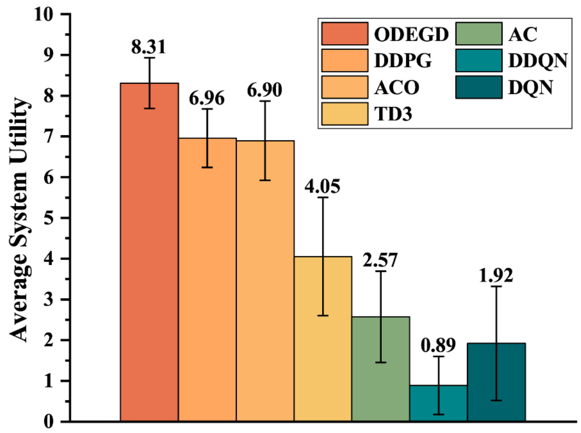
the optimal action produced by ACO. Although the training curve of the AC algorithm shows a convergence trend, the fluctuation amplitude of the confidence interval is large, which may be attributed to the instability caused by its use of real-time updates and hard updates. The ODEGD, DDPG, and TD3 algorithms have all converged, yet TD3's convergence is not optimal and exhibits a large fluctuation amplitude. The TD3 algorithm's complexity may make it less effective in solving this problem. DDPG converges within 200 episodes, which is faster than ODEGD at 600 episodes; however, its convergence results are not as robust as those of ODEGD. Fig. 8(b) presents the average system utility obtained by the aforementioned algorithm on the training set. It can be observed that ODEGD consistently yields the highest average system utility among all compared benchmark algorithms. Statistical analysis shows that ODEGD's 8.31 ± 0.62 significantly outperforms DDPG's 6.96 ± 0.72 (t(8) = 3.23, p = 0.012, 95% CI [0.38, 2.32], Cohen's d = 2.01) and ACO's 6.90 ± 0.97 (t(8) = 2.78, p = 0.023, 95% CI [0.24, 2.58], Cohen's d = 1.71). This is because our algorithm can provide the optimal movement path and optimal offloading rate during the online decision-making process, reducing the delay and energy consumption caused by task transmission and computation, thereby achieving the optimal system utility.

Fig. 9 presents the optimal trajectory provided by different algorithms for the UAV in the training set, with all algorithms set to depart from the same start point (500, 500, 100). From the perspective of the UAV's movement, DQN has been moving along the boundary, and there has been no height reduction in the early stage of simulation; DDQN continued to hover around the starting point and only lowered its altitude at the last moment; Although AC can reduce altitude, it makes almost no movement along the X-axis; ACO tends to decrease altitude and approach the central region, but it struggles to effectively search for the best flight trajectory. TD3 quickly lowers the UAV's altitude but, like AC, fails to further optimize along the Y-axis. The DDPG and ODEGD algorithms can make reasonable movements based on the UAV's 3D flight capability. From another perspective of the optimal offloading areas, AC and TD3 can allow agents to enter their respective optimal offloading areas, but these areas are too far away from the UEs' hotspot concentration area. Although the optimal area of DDPG is closer to the aggregation center of UE than ODEGD, it ignores UEs on remote roads.

Fig. 10 shows the strategy of optimal task offloading rate given by different DRL algorithms. We evaluate the performance of the algorithms in resource decision-making from the perspectives of each time slot and each UE. The average offloading rate across each time slot t, as plotted in Fig. 10(a), shows ODEGD's offloading rate is not the highest in the first two time slots. This is attributed to the distance between the UAV and the UE being too far in the early stage, which leads to higher transmission costs when attempting higher offloading rates. However, from the 3rd time slot onwards, ODEGD consistently



(a) Training curves



(b) Average system utility

Fig. 8. Training curves and average system utility with different optimization algorithms.

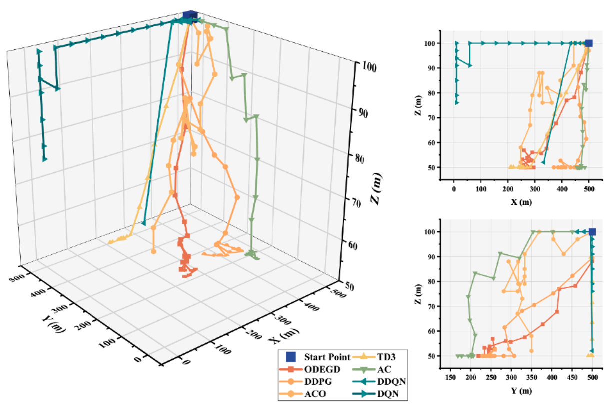
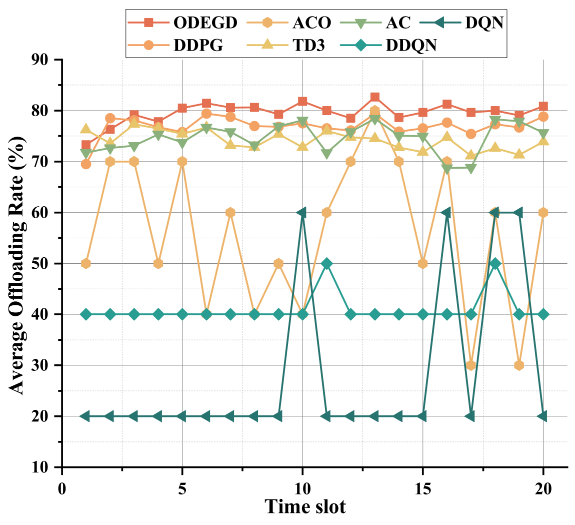
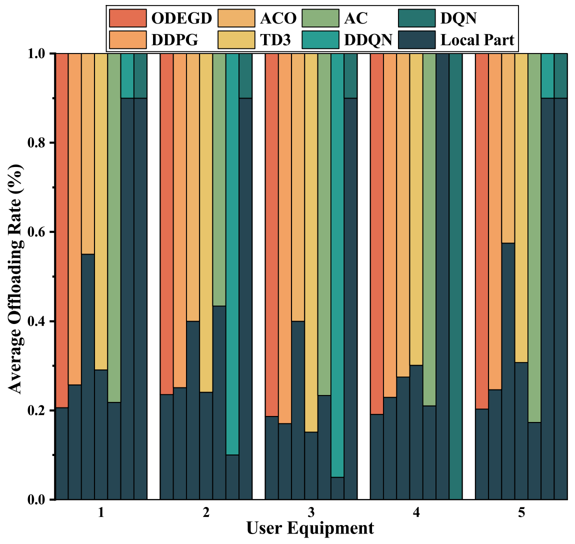


Fig. 9. UAV's 3D trajectory with different optimization algorithms.



(a) Each time slot t



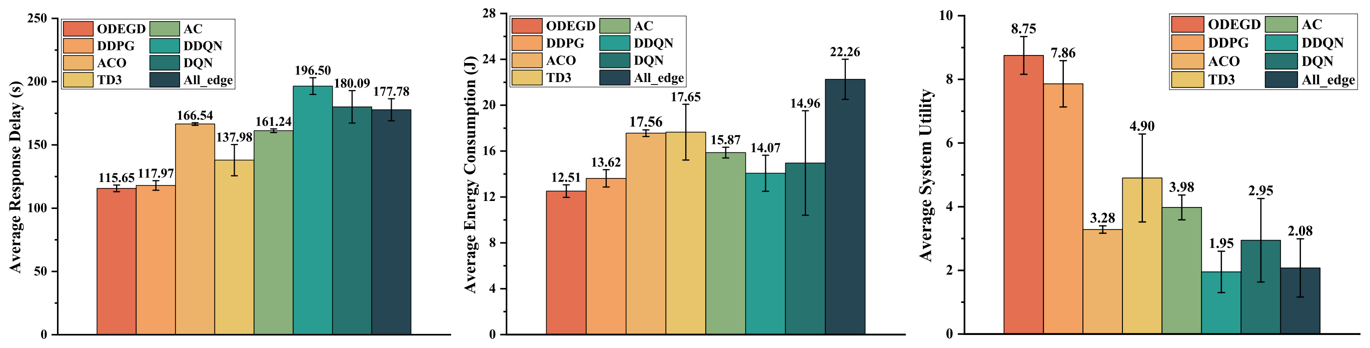
(b) Each UE m

Fig. 10. Average task offloading rate for UEs given by different optimization algorithms.

maintains the highest offloading rate, thereby accelerating task processing speed. ACO shows the highest variability in offloading rate, making it difficult to achieve stable performance. Additionally, TD3, DDPG, and AC also maintain relatively high offloading rates, while DQN and DDQN are quite low. The average offloading rate across each UE m is shown in Fig. 10(b). It can be seen that there is a certain gap in the offloading amount for different UEs among the comparative algorithms, with even cases where specific user tasks are not uninstalled at all, i.e., DDQN does not offloading tasks for UE 4. In contrast, ODEGD's offloading rate for different UEs is more balanced and stable, due to its strategy, which can efficiently support remote UEs while serving UEs in hotspot areas. The ODEGD algorithm maintains an average offloading volume of 79.6% across all simulation episodes, outperforming DDPG's 76.9%, ACO's 56%, TD3's 74.2%, AC's 74.6%, DDQN's 41%, and DQN's 28%. Therefore, ODEGD optimizes the resource allocation problem, significantly reducing the response delay and energy consumption for UEs' tasks from a computational models perspective.

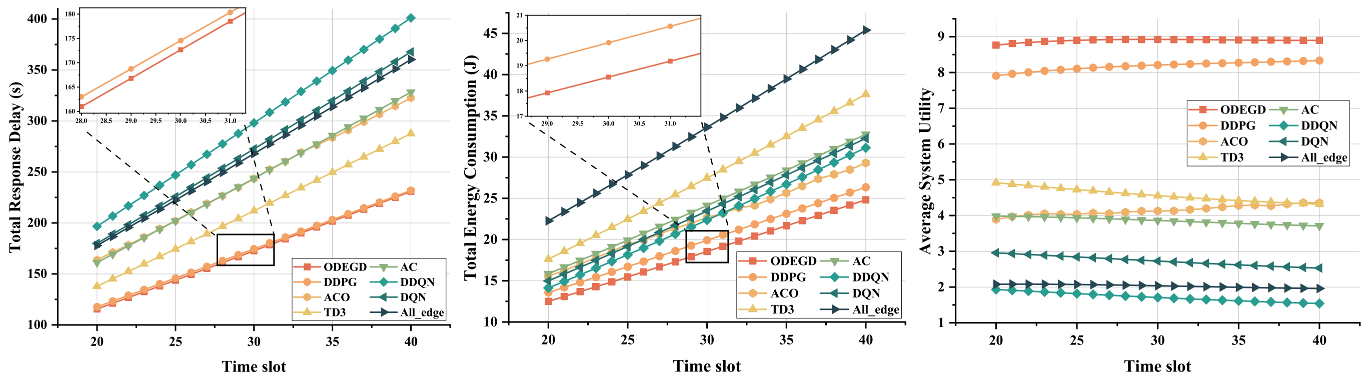
It is noteworthy that, in terms of average offloading rate, TD3 achieves 74.2%, which is comparable to AC's 74.6%, while DDQN's 41% exceeds DQN's 28%. Nevertheless, the average system utility results plots in Fig. 8(b) show that TD3's 4.05 is higher than AC's 2.57, and DQN's 1.92 exceeds DDQN's 0.89. This is because TD3 performs altitude descent earlier than AC in UAV's path planning, while DQN does not maintain hovering like DDQN. This indicates that the path planning of the UAV has a significant impact on system utility outcomes. Therefore, the algorithm needs to optimize both the UAV's path and resource allocation simultaneously to achieve higher system utility.

To further isolate and evaluate the effect of the proposed exploration strategy, we conducted an ablation study comparing ODEGD with



(a) Average response delay (b) Average energy consumption (c) Average system utility

Fig. 11. Performance comparison with different optimization algorithms.



(a) Total response delay (b) Total energy consumption (c) Average system utility

Fig. 12. Performance comparison with simulation time increase.

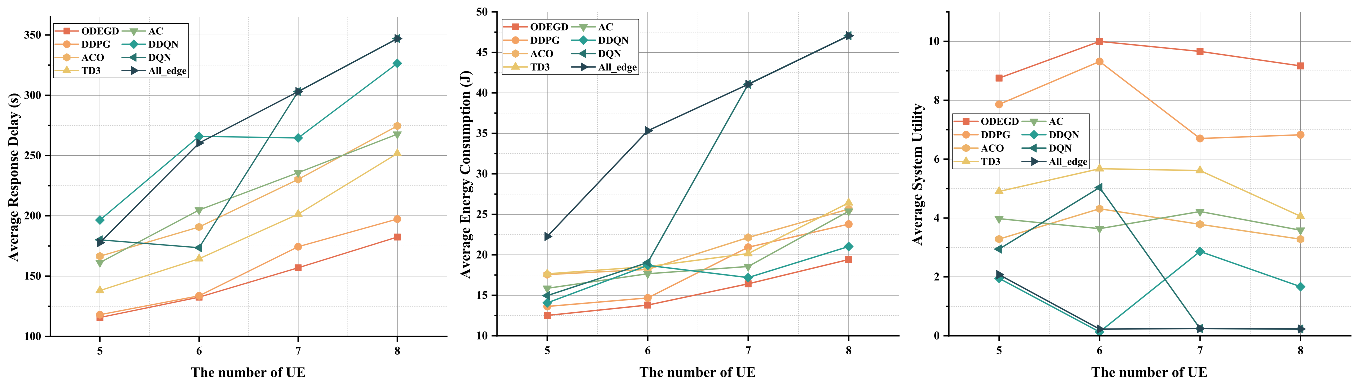
Table 3
Ablation study on the effect of the proposed exploration strategy.

Performance	ODEGD	DDPG + OU	DDPG + Gauss	DDPG + Greedy
Average response delay (s)	**115.65 ± 2.63**	117.97 ± 3.77	121.85 ± 3.11	119.22 ± 3.82
Average energy consumption (J)	**12.51 ± 0.55**	13.62 ± 0.76	14.18 ± 0.87	13.53 ± 0.67
Average system utility	**8.75 ± 0.59**	7.86 ± 0.73	7.27 ± 0.67	7.82 ± 0.66

several DDPG variants employing different exploration mechanisms, including DDPG + OU, DDPG + Gaussian, and DDPG + Greedy. The results are summarized in Table 3. As shown in the table, ODEGD consistently outperforms all methods across all performance metrics. Compared with DDPG + OU, ODEGD achieves a 1.97% reduction in Delay ($t(8) = 1.29$, $p = 0.23$, 95% CI [-1.57, 6.51], $d = 0.77$), a 8.14% reduction in Energy ($t(8) = 2.73$, $p = 0.026$, 95% CI [0.13, 2.08], $d = 1.63$), and an 11.36% improvement in Utility ($t(8) = 2.24$, $p = 0.054$, 95% CI [-0.03, 1.81], $d = 1.31$). The improvements are even more pronounced when compared to DDPG + Gaussian and DDPG + Greedy, with Utility gains of 20.38% and 11.88%, respectively. These results indicate that the proposed exploration strategy enables the agent to find an optimal policy more efficiently.

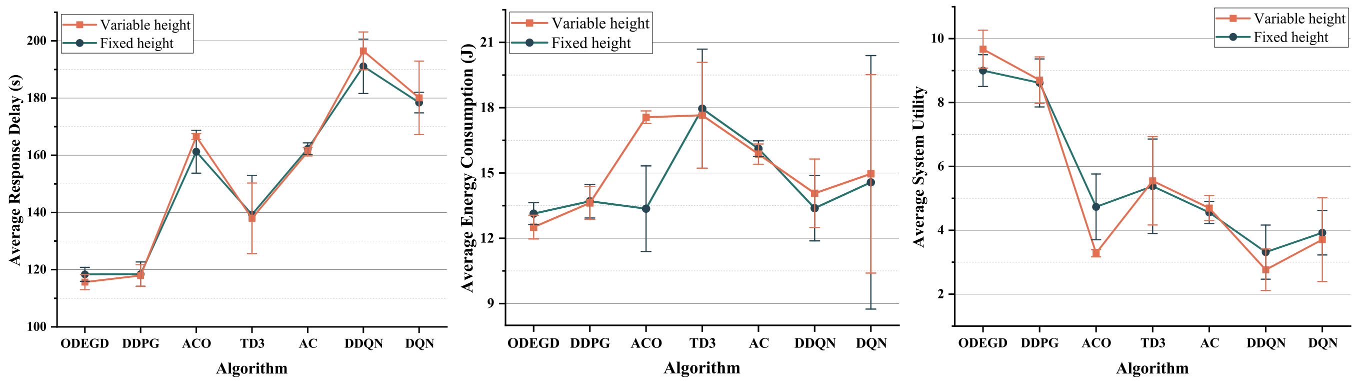
Fig. 11 indicates the performance of different algorithms in the testing set. To better compare the performance of algorithms, we have added a new method named offload all tasks to the edge (All_edge): During each test file, the UAV is deployed in a random place that is available and holds fixing. All UE’s tasks are offloaded to the edge server on the UAV throughout the process. From the Fig. 11, we can observe that the ODEGD algorithm achieves the lowest latency (115.65 ± 2.63 s) and energy consumption (12.51 ± 0.55 J) overhead, thus obtaining the highest system utility (8.75 ± 0.59). DDPG can also optimize the dual objectives of response delay and energy consumption, but its system utility (7.86 ± 0.73) is not as good as ODEGD. ACO shows a considerable performance drop on the test set compared to the training set. The reason is that its optimal solutions depend excessively on stochastic factors. Although it performs well in the training phase, large-scale testing exposes its insufficient ability to handle dynamic environments. TD3 and AC focus on reducing latency, leading to higher energy expenditure. Conversely, DDQN and DQN tend to reduce energy consumption, which results in longer task completion times. The All_edge method has the highest energy consumption, while there is no significant reduction in latency. Compared with other algorithms, ODEGD achieves a 26.98% reduction in average response delay and a 22.61% reduction in average energy consumption. Furthermore, it improves the average system utility by an average of 177.65%, with an increase of 78.57% compared to TD3 ($t(8) = 5.71$, $p < 0.001$, 95% CI [2.29, 5.41], Cohen’s $d = 3.70$), 166.77% compared to ACO ($t(8) = 20.26$, $p < 10^{-6}$, 95% CI [4.85, 6.09], Cohen’s $d = 11.10$), and 321.53% compared to All_edge ($t(8) = 13.23$, $p < 10^{-6}$, 95% CI [5.50, 7.84], Cohen’s $d = 9.05$).

Moreover, we extended the total simulation time and increased the UE counts to evaluate the scalability of each algorithm under increasing problem complexity. As shown in Fig. 12, the number of tasks increases with the extension of simulation time, so the total response delay and total energy consumption of the system will also increase. Throughout this process, the ODEGD algorithm consistently maintains the minimum latency and energy consumption. From the slope of curves plotted in Figs. 12(a), 12(b), it can be seen that ODEGD has the



(a) Average response delay (b) Average energy consumption (c) Average system utility

Fig. 13. Performance comparison with UE counts increase.



(a) Average response delay (b) Average energy consumption (c) Average system utility

Fig. 14. Performance comparison with different modes of height.

lowest growth rate in total response delay and total energy consumption. Fig. 12(c) shows the changes in the average system utility of different algorithms over time. It is noticeable that our proposed ODEGD algorithm achieved the highest system utility and remains stable, which is because after the UAV reaches the optimal offloading area, its path planning has a relatively small impact on the UAV's cost, and the system mainly focuses on resource allocation. The average system utility of DDPG shows an increasing trend because resource allocation issues become more important in the later stages of simulation, which DDPG can effectively address. However, the average system utility of other algorithms is very low and even shows a declining trend. This indicates that our proposed ODEGD algorithm can optimize both paths and resources, thereby maintaining the maximum utility in long-term simulations.

As shown in Fig. 13, when the number of UEs increases, the average response delay and average energy consumption of all algorithms show an overall upward trend due to the increased computation and communication load. From Figs. 13(a) and 13(b), it can be observed that ODEGD maintains the minimum delay and energy consumption with the slowest growth rate, reflecting its strong scalability in multi-user environments. Fig. 13(c) presents the variation of average system utility under different numbers of UEs. The ODEGD algorithm maintains the highest utility and strong stability even as the system load rises. The DDPG algorithm achieves relatively high utility but declines slightly when the UE number becomes large, suggesting reduced robustness under heavy loads. Other algorithms, such as DQN, DDQN, and ACO, show significantly lower system utility and poor adaptability as the number of users increases. These results demonstrate that ODEGD can effectively balance delay, energy consumption, and overall system performance, maintaining optimal utility even in complex multi-user environments.

To further evaluate the performance improvement of the ODEGD algorithm in solving the 3D trajectory optimization problem of the UAV, we evaluate the performance of different movement modes, as shown in Fig. 14. We assume that the UAV has two modes of movement: variable height flight mode and fixed height flight mode. From Fig. 14, we can observe that the average system utility of ACO, DDQN, and DQN decreased because these algorithms did not descend in altitude to reduce the distance between the UAV and UEs; instead, they ascended in altitude, leading to increased latency and energy consumption. The ODEGD, DDPG, TD3, and AC can all optimize the system's average latency and energy consumption by controlling the UAV's flight altitude, indicating that the UAV's 3D movement can further improve the offloading efficiency of the MEC system compared to the UAV's 2D movement. However, it can be seen from the figure that algorithms other than ODEGD have little effect on improving system utility, suggesting that these algorithms are not sufficient to solve the optimization problem with UAV's 3D continuous motion. The combination of ODEGD with variable height flight mode achieved the highest average system utility, improving by 7.47% compared to the fixed height flight mode ($t(8) = 1.96$, $p = 0.078$, 95% CI [-0.12, 1.46], Cohen's $d = 1.19$). This demonstrates the feasibility of our proposed ODEGD algorithm in solving large solution space problems.

6. Conclusion

To effectively address the partially observable and dynamic environment, we reformulated the optimization as a POMDP and developed ODEGD, an enhanced DRL-based online decision-making algorithm. ODEGD integrates a greedy exploration mechanism and observation-state approximation, enabling stable and robust learning in complex non-convex optimization problems. Incorporating real-world road data into our simulation experiments, we demonstrated the algorithm's effectiveness in practical scenarios. The simulation results show that our proposed method improves system utility by accurately perceiving task arrival patterns and UEs' movement trends. Our algorithm demonstrates superior performance over other optimization methods with respect to delay, energy consumption, and system utility. This study contributes a scalable and adaptive framework for continuous task offloading in realistic environments. For subsequent research, we plan to apply this framework to multi-UAV cooperative offloading and explore multi-objective optimization under practical network constraints.

CRediT authorship contribution statement

Zhao Tong: Writing – review & editing, Investigation, Funding acquisition, Conceptualization; **Shiyan Zhang:** Writing – original draft, Validation, Methodology, Investigation, Formal analysis, Data curation; **Jing Mei:** Writing – review & editing, Supervision; **Can Wang:** Writing – review & editing, Validation, Supervision, Investigation; **Keqin Li:** Writing – review & editing, Supervision.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

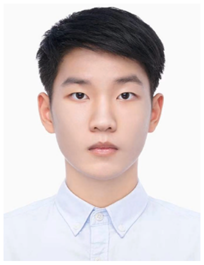
This work was supported by the Program of National Natural Science Foundation of China (Grant No. grantnumGS50110000180962372172, 62072174), Distinguished Youth Science Foundation of Hunan Province, China (Grant No. 2023JJ10030), National Natural Science Foundation of Hunan Province, China (Grant No. 2022JJ40278), Research Foundation of Education Bureau of Hunan Province, China (Grant No. 23B0100), Graduate Research Innovation Program of Hunan Province, China (Grant No. LXBZZ2024086).

References

[1] S. Xu, X. Zhang, C. Li, D. Wang, L. Yang, Deep reinforcement learning approach for joint trajectory design in multi-UAV IoT networks, IEEE Trans. Veh. Technol. 71 (3) (2022) 3389–3394.
[2] M. Sun, X. Xu, X. Qin, P. Zhang, AoI-Energy-Aware UAV-Assisted Data collection for IoT networks: a deep reinforcement learning method, IEEE Internet Things J. 8 (24) (2021) 17275–17289.
[3] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: the communication perspective, IEEE Commun. Surveys Tuts. 19 (4) (2017) 2322–2358.
[4] P.A. Apostolopoulos, G. Fragkos, E.E. Tsiropoulou, S. Papavassiliou, Data offloading in UAV-Assisted multi-Access edge computing systems under resource uncertainty, IEEE Trans. Mob. Comput. 22 (1) (2023).
[5] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam, M. Debbah, A tutorial on UAVs for wireless networks: applications, challenges, and open problems, IEEE Commun. Surveys Tuts. 21 (3) (2019) 2334–2360.
[6] S. He, S. Zhang, Trajectory planning in UAV-Assisted wireless networks via reinforcement learning, in: 2022 IEEE 23Rd International Conference on High Performance Switching and Routing (HPSR), 2022, pp. 232–237.
[7] Y. Gao, X. Yuan, D. Yang, Y. Hu, Y. Cao, A. Schmeink, UAV-Assisted MEC System with mobile ground terminals: DRL-Based joint terminal scheduling and UAV 3D trajectory design, IEEE Trans. Veh. Technol. 73 (7) (2024) 10164–10180.
[8] Y. Li, W. Liang, W. Xu, Z. Xu, X. Jia, Y. Xu, H. Kan, Data collection maximization in IoT-Sensor networks via an energy-Constrained UAV, IEEE Trans. Mob. Comput. 22 (1) (2023) 159–174.
[9] Y. Ma, Y. Tang, Z. Mao, D. Zhang, C. Yang, W. Li, Energy-Efficient 3D trajectory optimization for UAV-Aided wireless sensor networks, in: GLOBECOM 2023 - 2023 IEEE Global Communications Conference, IEEE, Kuala Lumpur, Malaysia, 2023, pp. 6591–6596.
[10] G. Chen, C. Cheng, X. Xu, Y. Zeng, Minimizing the age of information for data collection by cellular-Connected UAV, IEEE Trans. Veh. Technol. 72 (7) (2023) 9631–9635.
[11] Y. Zhu, B. Yang, M. Liu, Z. Li, UAV Trajectory optimization for large-Scale and low-Power data collection: an attention-Reinforced learning scheme, IEEE Trans. Wireless Commun. 23 (4) (2024) 3009–3024.
[12] J. Dandapat, N. Gupta, S. Agarwal, S. Darshi, Service duration maximization for continuous coverage in UAV-Assisted communication system, IEEE Commun. Lett. 26 (10) (2022) 2445–2449.
[13] Y. Lu, H. Zhou, H. Wang, T. Jiang, V.C.M. Leung, Online trajectory optimization and resource allocation in UAV-Assisted NOMA-MEC systems, in: 2024 IEEE/ACM 32Nd International Symposium on Quality of Service (IWQoS), 2024, pp. 1–2.
[14] Y. Guo, S. Yin, J. Hao, Resource allocation and 3-D trajectory design in wireless networks assisted by rechargeable UAV, IEEE Wireless Commun. Lett. 8 (3) (2019) 781–784.
[15] X. Qin, Z. Song, T. Hou, W. Yu, J. Wang, X. Sun, Joint optimization of resource allocation, phase shift, and UAV trajectory for energy-Efficient RIS-Assisted UAV-Enabled MEC systems, IEEE Trans. on Green Commun. Netw. 7 (4) (2023) 1778–1792.
[16] W. Zhao, S. Cui, W. Qiu, Z. He, Z. Liu, X. Zheng, B. Mao, N. Kato, A survey on DRL-Based UAV communications and networking: DRL fundamentals, applications and implementations, IEEE Commun. Surveys Tuts. 28 (2026) 3911–3941.
[17] K. Liu, J. Zheng, UAV Trajectory planning with interference awareness in UAV-Enabled time-Constrained data collection systems, IEEE Trans. Veh. Technol. 73 (2) (2024).
[18] X. Jiang, Z. Wu, Z. Yin, W. Yang, Z. Yang, Trajectory and communication design for UAV-Relayed wireless networks, IEEE Wireless Commun. Lett. 8 (6) (2019) 1600–1603.
[19] M. Masuduzzaman, A. Islam, K. Sadia, S.Y. Shin, UAV-Based MEC-assisted automated traffic management scheme using blockchain, Future Gener. Comput. Syst. 134 (2022) 256–270.
[20] J. Zhang, M. Lou, L. Xiang, L. Hu, Power cognition: enabling intelligent energy harvesting and resource allocation for solar-powered UAVs, Future Gener. Comput. Syst. 110 (2020) 658–664.
[21] B. Wang, Q. Wang, N. Yang, R. Chai, Long-Term optimization-Based data scheduling and trajectory planning for UAV-Assisted systems, in: 2023 IEEE 98Th Vehicular Technology Conference (VTC2023-Fall), IEEE, Hong Kong, Hong Kong, 2023, pp. 1–5.
[22] C. You, R. Zhang, 3D Trajectory optimization in rician fading for UAV-Enabled data harvesting, IEEE Trans. Wireless Commun. 18 (6) (2019) 3192–3207.
[23] J. Xiong, H. Guo, J. Liu, Task offloading in UAV-Aided edge computing: bit allocation and trajectory optimization, IEEE Commun. Lett. 23 (3) (2019) 538–541.
[24] X. Qin, Z. Song, Y. Hao, X. Sun, Joint resource allocation and trajectory optimization for multi-UAV-Assisted multi-Access mobile edge computing, IEEE Wireless Commun. Lett. 10 (7) (2021).
[25] D. Wang, J. Tian, H. Zhang, D. Wu, Task offloading and trajectory scheduling for UAV-Enabled MEC networks: an optimal transport theory perspective, IEEE Wireless Commun. Lett. 11 (1) (2022) 150–154.
[26] R. Chai, Y. Gao, R. Sun, L. Zhao, Q. Chen, Time-Oriented joint clustering and UAV trajectory planning in UAV-Assisted WSNs: leveraging parallel transmission and variable velocity scheme, IEEE Trans. Intell. Transport. Syst. 24 (11) (2023) 12092–12106.
[27] Z. Yang, S. Bi, Y.-J.A. Zhang, Online trajectory and resource optimization for stochastic UAV-Enabled MEC systems, IEEE Trans. Wireless Commun. 21 (7) (2022) 5629–5643.
[28] K.K. Nguyen, T.Q. Duong, T. Do-Duy, H. Claussen, L. Hanzo, 3D UAV Trajectory and data collection optimisation via deep reinforcement learning, IEEE Trans. Commun. 70 (4) (2022) 2358–2371.
[29] W. Zhao, P. Gao, X. Hong, X. Zheng, N. Kato, PPO-Based Task offloading with EKF for position prediction in RSU-Assisted IoV, IEEE Trans. Cogn. Commun. Netw. 12 (2026) 703–713.
[30] H. Hu, K. Xiong, G. Qu, Q. Ni, P. Fan, K.B. Letaief, AoI-Minimal Trajectory planning and data collection in UAV-Assisted wireless powered IoT networks, IEEE Internet Things J. 8 (2) (2021) 1211–1223.
[31] J. Zhang, Y. Zeng, R. Zhang, UAV-Enabled Radio access network: multi-Mode communication and trajectory design, IEEE Trans. Signal Process. 66 (20) (2018) 5269–5284.
[32] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, D. Niyato, Multi-Agent deep reinforcement learning for task offloading in UAV-Assisted mobile edge computing, IEEE Trans. Wireless Commun. 21 (9) (2022) 6949–6960.
[33] M. Coldrey, J.E. Berg, L. Manholm, C. Larsson, J. Hansryd, Non-Line-of-Sight small cell backhauling using microwave technology, IEEE Commun. Mag. 51 (9) (2013) 78–84.
[34] Z. Tong, J. Wang, J. Mei, K. Li, K. Li, Fedto: mobile-Aware task offloading in multi-Base station collaborative MEC, IEEE Trans. Veh. Technol. 73 (3) (2024) 4352–4365.
[35] Z. Tong, J. Wang, J. Mei, K. Li, W. Li, K. Li, Multi-type task offloading for wireless internet of things by federated deep reinforcement learning, Future Gener. Comput. Syst. 145 (2023) 536–549.
[36] Z. Yu, Y. Guo, Joint task offloading and resource allocation in UAV-Enabled mobile edge computing, IEEE Internet Things J. 7 (4) (2020).

[37] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, Computer ence (2015).
[38] S. Li, F. Wu, S. Luo, Z. Fan, J. Chen, S. Fu, Dynamic online trajectory planning for a UAV-Enabled data collection system, IEEE Trans. Veh. Technol. 71 (12) (2022) 13332–13343.
[39] A. Keränen, J. Ott, T. Kärkkäinen, The ONE simulator for DTN protocol evaluation, in: Proceedings of the Second International ICST Conference on Simulation Tools and Techniques, ICST, Rome, Italy, 2009.
[40] Y. Wang, W. Fang, Y. Ding, N. Xiong, Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach, Wireless Netw. 27 (4) (2021) 2991–3006.

Zhao Tong is currently an professor with Hunan Normal University. He was a visiting scholar at the Georgia State University during 2017–2018. He has author or coauthored more than 50 papers in peer-reviewed international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Services Computing, IEEE Transactions on Network and Service Management, IEEE Transactions on Vehicular Technology, etc. His research interests include AI computing, parallel and distributed computing systems, and resource management. He is a senior member of the IEEE and CCF.


Shiyan Zhang received his bachelor’s degree from Guangxi Minzu University, Nanning, China, in 2022. He is currently pursuing his master’s degree in the College of Information Science and Engineering of Hunan Normal University, Changsha, China. His research interests mainly revolve around the areas of UAV-assisted mobile edge computing and deep reinforcement learning.

Jing Mei received the Ph.D. degree in computer science from Hunan University, China, in 2015. She is currently an associate professor at the Hunan Normal University, Changsha, China. Her research interests include parallel and distributed computing and cloud computing. She has published 40 research articles in international conference and journals, such as IEEE-TC/TPDS/TSC/Cluster Comput./J. Grid Comput./J. Supercomput, etc.

Can Wang received the B.E. degree from Tianjin Normal University, Tianjin, China, in 2023. She is currently pursuing the M.S. degree with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests focus on mobile edge computing, task offloading and objective optimization.

Keqin Li is currently a SUNY Distinguished Professor of computer science. He has authored or coauthored more than 510 journal articles, book chapters, and refereed conference papers. His research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems. He was the recipient of the several best paper awards. He is on the Editorial boards of the IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, and IEEE Transactions on Sustainable Computing.