Reliability-aware Optimization of Task Offloading for UAV-assisted Edge Computing

Hao Hao, Changqiao Xu, *Senior Member IEEE,* Wei Zhang, Xingyan Chen, Shujie Yang, and Gabriel-Miro Muntean, *Fellow IEEE*

***Abstract*—Unmanned aerial vehicles (UAV) are widely used for edge computing in poor infrastructure scenarios due to their deployment flexibility and mobility. In UAV-assisted edge computing systems, multiple UAVs can cooperate with the cloud to provide superior computing capability for diverse innovative services. However, many service-related computational tasks may fail due to the unreliability of UAVs and wireless transmission channels. Diverse solutions were proposed, but most of them employ time-driven strategies which introduce unwanted decision waiting delays. To address this problem, this paper focuses on a task-driven reliability-aware cooperative offloading problem in UAV-assisted edge-enhanced networks. The issue is formulated as an optimization problem which jointly optimizes UAV trajectories, offloading decisions, and transmission power, aiming to maximize the long-term average task success rate. Considering the discrete-continuous hybrid action space of the problem, a dependence-aware latent-space representation algorithm is proposed to represent discrete-continuous hybrid actions. Furthermore, we design a novel deep reinforcement learning scheme by combining the representation algorithm and a twin delayed deep deterministic policy gradient algorithm. We compared our proposed algorithm with four alternative solutions via simulations and a realistic Kubernetes testbed-based setup. The test results show how our scheme outperforms the other methods, ensuring significant improvements in terms of task success rate.**

***Index Terms*—UAV-assisted network, multi-access edge computing (MEC), deep reinforcement learning (DRL).**

I. Introduction

With the rapid development of smart mobile devices, computation-intensive services such as virtual reality (VR), augmented reality (AR) and extended reality (XR) become increasingly popular, greatly enriching people's daily life, but also increasing computing and communication demands. Cloud computing which offloads all tasks to remote cloud servers can realize the centralization of storage, computing and network management. However, it is impractical to meet all the increasing computing demands by relying only on the computing resources of the cloud server. Additionally, cloud computing is usually associated with high transmission delays due to the long transmission distance between remote users and the cloud server, which makes difficult to guarantee high performance of some delay-sensitive tasks. To address the limitations of cloud computing, a new computing paradigm named Multi-access Edge Computing (MEC) has emerged [1]. MEC provides computational functions at the network edge to achieve high bandwidth and low latency, which is very useful especially for computation-intensive services.

However, there are still bottlenecks in achieving high-quality computation-intensive services in many critical emergency scenarios, such as emergency management and disaster relief response. The reason is that in the existing edge computing architecture, edge servers are usually embedded in fixed ground infrastructure such as base stations (BSs) and access points (APs). On one hand, it is difficult to deal with a large number of temporary burst computing tasks in rural areas with little ground infrastructure or during peak hours in urban areas [2]. On the other hand, natural disasters or military strikes could potentially cause damage to the ground infrastructure, leading to an inability to support rescue, testing and other services [3].

Leveraging their flexibility and mobility, unmanned aerial vehicles (UAVs) can extend the wireless networks' support by offering services to ground users, including data collection, rapid network access, and edge computing [4]. UAVs can be used as aerial servers to provide computing support by dynamically increasing system's computing resources, offer better support for on-demand computing tasks, and solve more efficiently problems related to insufficient computing power in the user proximity, especially in emergency scenarios. It is a promising solution for MEC to provide flexible computational services by strategically deploying edge support on UAVs.

In an UAV-assisted MEC network, UAVs can function as BSs, relays, or servers, to improve system performance [5]. Unlike stationary ground BSs, UAVs possess agile mobility, allowing them to approach ground users closely, thereby establishing high-throughput Line-of-Sight (LoS) connections. Deploying MEC-capable UAVs not only facilitates user computation offloading and edge service deployment simultaneously, but also can reduce the costs associated with constructing communication and computing infrastructures. However, the dynamic mobility of UAVs introduces challenges in their effective deployment and coordination, especially in scenarios involving multiple UAVs. In addition, the onboard computing

H. Hao, W. Zhang are with Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, China, and with Shandong Provincial Key Laboratory of Computer Networks, Shandong Fundamental Research Center for Computer Science, Jinan, China (e-mail: haoh@sdas.org, wzhang@sdas.org).

C. Xu, S. Yang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. (e-mail: cqxu@bupt.edu.cn, sjyang@bupt.edu.cn).

X. Chen is with the School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu, China (e-mail:xychen@swufe.edu.cn).

G.-M. Muntean is with the Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University, Dublin, Ireland. (e-mail: gabriel.muntean@dcu.ie).

Corresponding Author: Shujie Yang

resources and energy supply of UAVs are quite limited, making very challenging any related edge computing decision [6]. Therefore, realizing a joint optimization of the UAVs' trajectory, offloading decision, and communication resources allocation has become an urgent issue to address in the UAV-assisted MEC network.

Most of the existing works proposed time-driven schemes, which means that computing decisions need to be made at the end of each time slot [7]. The time slot of the actual system clock is much shorter than the time slot of task offloading in the time-driven work. For example, the duration of a time slot in a network switch is usually measured in microseconds (us) or nanoseconds (ns), while the time slot is usually set to 10ms-100ms in task offloading models. On one hand, the time slot in task offloading models cannot be set as small as the system clock's time slot. As wireless channels serve as the connection medium between user equipments (UEs) and UAVs, there is usually a strong association between the time slot and the coherence time of the wireless channel in a wide range of application cases [8][9]. On the other hand, offloading decisions have to be made frequently if time slot intervals are set too short, which consumes a lot of computing resources. Besides, most task offloading algorithms are difficult to be performed in microseconds. Therefore, the existing time-driven task offloading algorithms cannot avoid the decision waiting delay caused by the different time slot granularity. In addition, most of the literature on UAV-assisted MEC networks takes minimizing the task delay as the optimization objective. However, the UAV itself or the transmission link between UAVs and UEs are all unstable, which may fail while processing or transmitting tasks. In addition, most tasks will satisfy users' demand as long as they are completed before their deadlines. Blindly optimizing delay cannot greatly improve the quality of service (QoS), but improving task success rate while considering system reliability is a key avenue.

In this context, we propose a Task-driven Reliability-aware Offloading (TRO) scheme for a UAV-assisted MEC network. First, we construct a system reliability model, and formulate a joint optimization problem of UAVs' trajectory, offloading decision and communication resources, which is task-driven. Secondly, considering that the optimization variables are composed of discrete and continuous values, we propose a dependence-aware latent representation algorithm to help. Finally, we design a novel Deep Reinforcement Learning (DRL) algorithm to solve the problem without any prior knowledge. The main contributions of the paper are summarized as follows:

- We propose the task success rate model, and formulate a joint optimization problem with the objective of maximizing the long-term average task success rate in UAV-assisted MEC network. The UAV mobility model and computation model are constructed from a task perspective. For a more realistic scenario, we also consider the reliability of UAVs and transmission links.
- We further transform the optimization problem into a Markov decision process (MDP). Considering a hybrid action space with discrete and continuous variables, we propose a dependence-aware latent representation algorithm. Then, combining the representation space with the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, we propose a novel DRL algorithm to solve this optimization problem.
- In order to quickly verify the effectiveness of our algorithm, we conduct a large number of simulation experiments. Moreover, we have also built a platform based on Kubernetes to realize data transmission in real scenarios, so as to validate the practicality and applicability of our proposed scheme. Experiments show that our algorithm can achieve better performance than other four state-of-the-art algorithms.

II. Related Works

UAVs have flexibility and mobility advantages, which can help in terms of rapid deployment and on-demand allocation of edge computing resources. UAV-assisted MEC networks have attracted the attention of many scholars.

Multiple articles have focused on single UAV-assisted MEC scenarios. The authors of [10] studied the system throughput maximization problem for a single UAV-to-community offloading system. They proposed an average throughput maximization-based auction algorithm to design the trajectory of UAV, and developed an algorithm to determine the task scheduling policy. Focusing on the energy consumption of UAV and security of data transmission, the authors of [11] transformed the energy-efficient offloading problem into a convex problem. Then, the authors proposed an optimal solution through strict mathematical proofs. To minimize the task delay and energy consumption simultaneously, the authors of [12] formulated the joint optimization problem of UAV's trajectory, resources allocation and task offloading decisions. Considering that the problem was highly nonconvex, they proposed an algorithm based on successive convex approximations to obtain suboptimal solutions. Although considering a single UAV with fewer computing resources simplifies the optimization problem, it is difficult to meet the increasing demands for service resources and complex missions.

For multiple UAV-assisted scenarios, several research works have proposed their own algorithms based on conventional optimization methods. The authors of [13] formulated a UAV-assisted MEC problem aiming to minimize the total energy consumption, with constraints of task delay and dependency. To solve this problem, they decomposed original problem into two subproblems based on block coordinate descent, and solved these subproblems by dynamic programming and convex optimization methods. The authors of [14] proposed a cooperative multiple UAV-assisted edge computing system based on software defined network. They studied the problem of joint task offloading and computing resource allocation. Then a two-layer game-theoretic approximation was proposed to balance energy consumption of UAVs and minimize the task delay. The authors of [15] designed an UAV-enabled multi-hop collaborative system model, where multiple UAVs provided effective computation services for UEs. They formulated their problem as a mixed integer nonlinear programming (MINLP)

problem and designed a multi-hop collaborative algorithm to solve it. Some more aspects corresponding to the practical systems (e.g. the stochastic arrival of practical tasks, network congestion) were considered in [16]. The authors studied a task delay minimization problem for multi-UAV assisted cooperative offloading, and formulated it as a non-convex problem. The UAV was introduced to address the vehicular edge computing overload problem in [17]. The authors constructed an optimization problem with the long-term UAV energy consumption as constraint and minimization of the vehicular task delay, as the optimization objective. They first decoupled the long-term energy constraint based on Lyapunov stochastic optimization and then found close-to-optimal solution by Markov approximation optimization. Nevertheless, the computational complexity of conventional optimization methods increases sharply as the number of UAVs or UEs increases [18].

Through exploration of the dynamic environments and training on historical experience, DRL has emerged as an excellent alternative for optimizing dynamic large-scale problems. Many research works have introduced DRL to multi-UAV MEC system. The authors of [19] proposed an offline-to-online DRL algorithm to achieve task offloading optimization of multi-node cooperation. The approach can also build upon prior heuristic algorithms enhancing their performance. In [18], each UAV jointly optimized energy consumption of itself and UEs, the stability of task queues, by designing the trajectory and task offloading ratio. The authors formulated the problem and transformed it using a Lyapunov stochastic optimization. Then, a DRL-based algorithm was proposed and compared with other algorithms. In [20], the authors formulated the joint optimization of UAVs' trajectory, task offloading, resource allocation as a mixed integer programming problem, which highlighted task priorities. To solve the problem, they proposed a novel DRL-based algorithm. The authors of [21] aimed to simultaneously minimize the short-term computational costs of UEs and the long-term computational cost of UAVs, based on incomplete information in an UAV-assisted MEC network. They first formulated the problem as a Markov game model and proposed a DRL-based optimization algorithm to solve it. The authors in [22] formulated task offloading in an UAV-assisted MEC network as a multi-objective optimization, whose goals are to minimize task delay, minimize system energy consumption, and maximize the number of tasks collected by UAVs, simultaneously. They proposed a multi-objective DRL algorithm to solve it.

Previous works have achieved excellent results, but several critical design aspects have bot been adequately addressed. First, most of the works on task offloading for the UAV-assisted MEC network are time-driven, and a task-driven strategy has not been studied in the literature. In some scenarios (e.g., computation-intensive task computation), time-driven schemes often result in decision waiting delays or frequent offloading decisions, while task-driven algorithm has smaller implementation complexity and lower task delays. A task-driven algorithm may not be suitable to all cases, indeed, but it is worth studying. Secondly, task execution in real scenarios may fail due to the unreliability of wireless channels and

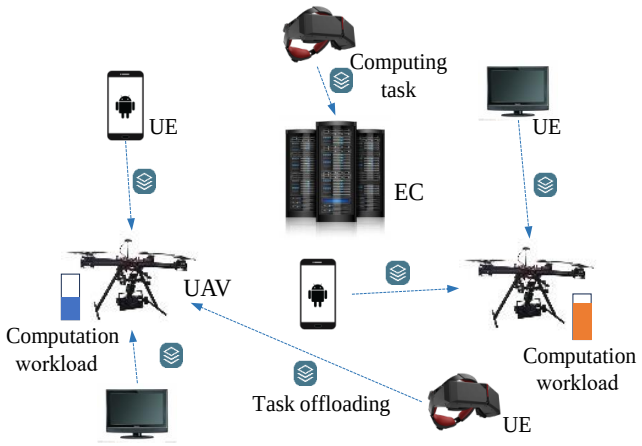


Fig. 1. Overview of system model

UAVs. It is unwise to ignore the reliability of a system and only optimize task delay or energy consumption, as reliability issues may cause some tasks to fail. It is important to optimize the task success ratio considering the reliability of system, which has not been addressed. In this paper, we will address these open issues by designing a task-driven reliability-aware offloading scheme for UAV-assisted MEC network.

III. System Model

In this section, we outline the system's core components, encompassing the network model, movement of UAVs, and reliability model.

A. Network Model

Consider a UAV-assisted MEC system comprising N UAVs, M UEs, and an edge cloud server (EC). UAVs offer computational resources to assist UEs in completing their computation tasks, as shown in Fig. 1. Each UAV is equipped with multiple antennas and can receive tasks from multiple UEs simultaneously [23]. Each UE $m \in \{1, 2, ..., M\}$ has the option to process tasks locally or offload them to any UAV $n \in \{1, 2, ..., N\}$ or EC. Each UAV n manages a queue for task execution, following a First-In, First-Out (FIFO) order. Tasks in the queue must await their turn if computational resources are occupied. The computation capacity of UAV n is denoted as F_n. Noteworthy is that the system is not completely decentralized. Decisions in terms of task offloading and UAV trajectories are controlled by the edge cloud server based on distributed information. This is as computing and energy resources in UAVs are limited. In a completely decentralized system, UAVs need to make all decisions, which will inevitably increase both their computational overhead and energy consumption, which is undesirable. Additionally, usually decisions cannot be made based on global information in a completely decentralized system and the accuracy of decisions based on local information is often lower than those based on global information.

The model is task-driven rather than time-driven, meaning we do not depict system evolution by increasing time slots. Instead, we describe it based on the arrival of computational tasks. There are two advantages of the task-driven model. First,

we can make offloading decisions as soon as the task arrives instead of the end of time slots, which can avoid the decision waiting delay caused by different time slot granularity in time-driven schemes. Second, we only need to make offloading decisions for one task at a time, unlike in time-driven models, where there is a need to determine the offloading status of all computational tasks at each time slot. The model complexity often scales exponentially with the dimension of the decision, and by making decisions on one task at a time, the model complexity reduces. Therefore, our task-driven scheme has lower implementation complexity and lower task delay, which is suitable for cases where the task generation frequency is low and there are low delay requirements.

Task k denotes the k-th computational task generated by UEs during system operation, rather than a specific computational task. For example, consider a device which can perform two computing tasks: model training and image coding. In timeslot-driven studies, these two tasks are represented by task 1 and task 2, and the offloading status of these two tasks needs to be determined at each time slot. In our task-driven model, there is no time slot division and tasks are generated repeatedly (e.g. model training, image coding, model training, image coding). To keep track of the tasks, we allocate them numbers based on the order in which they are generated (e.g. task 1, task 2, task 3 and task 4). Although task 1 and task 3 are both model training, as they are generated at different times, we will allocated them different task numbers. In this way, system evolution is described by the arrival of computational tasks rather than based on time slots. This approach has two advantages. First, we can trigger the task decision as soon as the task is generated, rather than waiting until the end of each time slot, which avoids introducing decision waiting delays. Second, the model complexity often scales exponentially with the dimension of the decision, and we only make decisions on one task at a time, reducing the model complexity. Such a task-driven model is most appropriate for scenarios with stringent latency requirements and low task frequencies, ensuring optimal performance.

We represent task k as a five-tuple $(c_k, u_k, t_k, m_k, d_k)$, where c_k denotes the computing workload (measured in CPU cycles), u_k represents the data transmission size, t_k indicates the arrival time (a continuous value), m_k identifies the UE generating task k, and d_k signifies the permissible delay threshold. The binary task offloading problem is studied, where the task is offloaded as a whole [24]. We introduce a multivariate variable $i_k \in \{0, 1, ..., N, N+1\}$ to denote the offloading destination of task k: $i_k = 0$ implies task k is processed locally by UE m_k, $i_k \in \{1, \ldots, N\}$ indicates task k is offloaded to UAV i_k, and task k is offloaded to the EC if $i_k = N + 1$.

B. UAVs Movement

We define the 3D position of UAV n at task k arrival as $\mathbf{w}_n(k) = [x_n(k), y_n(k), z_n(k)]^T$, where $x_n(k)$, $y_n(k)$ and $z_n(k)$ represent its respective X, Y, and Z coordinates. The 2D position $\mathbf{v}_n(k) = [x_n(k), y_n(k)]^T$ denotes the horizontal plane coordinates of UAV n. Due to the constrained horizontal and vertical speeds during flight, UAVs have limited travel distances, described as follows:

$$\triangle v_n(k) = |\mathbf{v}_n(k+1) - \mathbf{v}_n(k)| \leq L^h_{max} \cdot \triangle k \quad (1)$$

$$\triangle z_n(k) = |z_n(k+1) - z_n(k)| \leq L^v_{max} \cdot \triangle k \quad (2)$$

where $\triangle k$ is the interval duration between the arrival time of task k and $k+1$, L^h_{max} and L^v_{max} represent the maximum horizontal speed and maximum vertical speed UAV n can travel, $\triangle v_n(k)$ and $\triangle z_n(k)$ denote the horizontal and vertical travel distances of UAV n, respectively.

To prevent collisions between UAVs, the distance between any two UAVs must not fall below a minimum distance D_{min}. This collision constraint is expressed as follows:

$$||\mathbf{w}_n(k) - \mathbf{w}_j(k)|| \geq D_{min}, \quad if \quad n \neq j \quad (3)$$

During the flight of a UAV, the energy consumption power is correlated with the speed v [25]:

$$P^{fly}_n(v) = \frac{W_n}{2} v^2 \quad (4)$$

where W_n is the mass of UAV n. The flight energy consumption of UAV for task k is:

$$E^{fly}_n(k) = P^{fly}_n(\frac{||\mathbf{w}_n(k+1) - \mathbf{w}_n(k)||}{\triangle k}) \triangle k \quad (5)$$

C. Reliability Model

The reliability of system includes two aspects: transmission reliability and computing unit (such as UAVs) reliability. Failures in these components can be modeled by a Poisson distribution driven by a failure rate [26]. In this context, the processing time required by node $j \in \{0, 1, 2, ..., N+1\}$ to complete task k without failure is denoted as $t^{co}_{j,k}$. During the time interval $(0, t^{co}_{j,k}]$, failures are assumed to occur as a Poisson process with a failure rate parameter α_j. The total number of failures occurring in node j when processing task k is represented by $H_j(t^{co}_{j,k})$. Thus, the probability of $H_j(t^{co}_{j,k}) = h$ during the time interval $(0, t^{co}_{j,k}]$ can be calculated as:

$$Pr\{H_j(t^{co}_{j,k}) = h\} = \frac{(\alpha_j t^{co}_{j,k})^h}{h!} e^{-\alpha_j t^{co}_{j,k}}, h \geq 0 \quad (6)$$

where $h = 0$ indicates that task k is processed successfully during the time interval $(0, t^{co}_{j,k}]$. The reliability of task k processed by node j can be determined as:

$$R^c_{j,k} = e^{-\alpha_j t^{co}_{j,k}} \quad (7)$$

The reliability model is mainly for soft errors (e.g., unexpected restart/shutdown, communication errors and outages, software bugs, etc), but it can also consider hardware errors by adjusting the failure rate parameter. Hardware errors often cause equipment to be unavailable for a long time (e.g. damaged UAV). In this case, the system treats the UAV as offline and no more tasks are assigned to it. After the hardware failure, we can set the failure rate parameter in the fault model to a very large value, so that the reliability of the equipment is close to 0, and it will not be assigned any tasks. In the reliability model, we calculate reliability based on task execution time. For each task, we separately calculate the

probability that the equipment will not fail during execution. Notably, soft errors often can be automatically fixed through fault recovery techniques, such as roll forward/rollback or checkpoint. After the fault is rectified, the equipment can continue to operate. So we assume that the failure is a one-off incident that does not impact subsequent tasks [27]. For a hardware error, we can modify the failure rate parameter and set it to a very large value. In this way, the system will not assign tasks to the equipment affected, and this does not impact any subsequent tasks. Similarly, the reliability of task k when transferred from UE m to j can be computed as:

$$R^t_{m,j,k} = e^{-\beta_{m,j} t^{tr}_{m,j,k}} \tag{8}$$

where $\beta_{m,j}$ denotes the failure rate of the transmission link between UE m and node j.

IV. Problem Analysis and Formulation

In this section, we first construct models of local computing, edge computing and cloud computing, then formulate an optimization problem to maximize long-term task success. Finally, we transform it into a MDP.

A. Local Computing

If task k is processed at the UE locally, it should be added to the local computation queue. The update of computation queues can be categorized into two scenarios: the UE that generates task k, and other UEs.

For the UE m_k which generates task k, the update of its computation queue is as follows:

$$q_{m_k}(k) = [q_{m_k}(k-1) - (t_k - t_{k-1}) f^d_{m_k}]^+ + c_k \tag{9}$$

where $q_{m_k}(k)$ represents the computation queue of UE m_k upon the arrival of task k. The operator $[z]^+ = max\{0, z\}$ ensures non-negativity, $t_k - t_{k-1}$ denotes the interval between the arrival of task $k-1$ and task k, $(t_k - t_{k-1}) f^d_{m_k}$ represents the computed workload during this interval, $[q_{m_k}(k-1) - (t_k - t_{k-1}) f^d_{m_k}]^+$ represents the current workload of the UE m_k.

For other UEs ($m \neq m_k$), the computation queues are:

$$q_m(k) = [q_m(k-1) - (t_k - t_{k-1}) f^d_m]^+ \tag{10}$$

In summary, the update of computation queues of devices is expressed as:

$$q_m(k) = [q_m(k-1) - (t_k - t_{k-1}) f^d_m]^+ + \mathbf{1}_{\{m=m_k\}} c_k \tag{11}$$

where $\mathbf{1}_{\{z\}} = 1$ if condition z is true, otherwise $\mathbf{1}_{\{z\}} = 0$.

There is no transmission delay, hence the service delay corresponds to the local processing delay of task k:

$$T^l(k) = \frac{q_{m_k}(k)}{f^d_{m_k}} \tag{12}$$

where $f^d_{m_k}$ is the computing capability of UE m_k.

The reliability of task k processed locally is

$$R^{local}_k = R^c_{m_k,k} = e^{-\alpha_{m_k} t^{co}_{m_k,k}} \tag{13}$$

where $t^{co}_{m_k,k} = c_k / f^d_{m_k}$ is the computation time for the task k processed locally.

B. Edge Computing

For UAVs, the network is dynamic and the transmission rate is related to the position of UAVs. In UE-to-UAV communication links, the real environment often contains numerous obstacles or scatterers, leading to increased path loss. Therefore, employing a simplified free space path loss (FSPL) model [25] is inadequate. A probabilistic path loss model is proposed instead, as it takes into account the probabilities and path losses associated with Line-of-Sight (LoS) and Non-Line-of-Sight (NLoS) communications in UE-to-UAV scenarios. The occurrence probabilities are given by:

$$P^{LoS}_{m,n}(k) = \frac{1}{1 + a e^{-b((180/\pi) arcsin(z_n(k)/dis_{m,n}(k)) - a)}} \tag{14}$$

$$P^{NLoS}_{m,n}(k) = 1 - P^{LoS}_{m,n}(k) \tag{15}$$

where $P^{LoS}_{m,n}(k)$ is the LoS communications probability between UE m and UAV n, $P^{NLoS}_{m,n}(k)$ is the NLoS communications probability, $dis_{m,n}(k) = ||\mathbf{w}_n(k) - \mathbf{w}_m(k)||$ is the distance between UE m and UAV n, a and b are constant values, $arcsin(z_n(k)/dis_{m,n}(k))$ represents converting the ratio of the height and distance of the UAV into radians, $(180/\pi) arcsin(z_n(k)/dis_{m,n}(k)) - a$ represents the angle adjusted by the constant a, $a e^{-b((180/\pi) arcsin(z_n(k)/dis_{m,n}(k)) - a)}$ reflects the attenuation characteristics of the signal with distance and angle. The formula converts factors such as UAV signal characteristics, environmental conditions, and equipment performance into the probability of successful communication. The path loss between UE m and UAV n is modeled as follows:

$$PL^{\zeta}_{m,n}(k) = L_{m,n}(k) + \eta_{\zeta}, \quad \zeta \in \{LoS, NLoS\} \tag{16}$$

where $L_{m,n}(t) = 20 log_{10}(4\pi/c) + 20 log_{10}(fr_c) + 20 log_{10}(dis_{m,n}(k))$ is the free space path loss, fr_c is the carrier frequency, c is the speed of light, and η_{ζ} is excessive path loss of LoS or NLoS links. We get the average path loss for UE-to-UAV links next:

$$\bar{PL}_{m,n}(k) = PL^{LoS}_{m,n}(k) P^{LoS}_{m,n}(k) + PL^{NLoS}_{m,n}(k) P^{NLoS}_{m,n}(k) \tag{17}$$

The channel gain between UE m and UAV n is

$$g_{m,n}(k) = 1 / \bar{PL}_{m,n}(k) \tag{18}$$

The uplink transmission rate from UE m to UAV n is:

$$r^{U2A}_{m,n}(k) = B^A log_2(1 + \frac{p_m(k) g_{m,n}(k)}{N_A}) \tag{19}$$

where B^A is channel bandwidth for UE-to-UAV links, $p_m(k)$ is the transmit power of UE m, and N_A is the noise power. The transmission time is:

$$t^u(k) = \frac{u_k}{r^{U2A}_{m_k,i_k}(k)} \tag{20}$$

There are also computation queues in UAVs due to the constrained computing resources. Similar to UEs, the update of computation queues is as follows:

$$Q_n(k) = [Q_n(k-1) - (t_k - t_{k-1}) f^e_n]^+ + \mathbf{1}_{\{n=i_k\}} c_k \tag{21}$$

where $Q_n(k)$ is the computation queue of UAV n when task k arrives, $\mathbf{1}_{\{n=i_k\}}$ indicates whether UAV n is the offloading target or not.

The computing delay of edge computing for task k is:

$$t^b(k) = \frac{Q_{i_k}(k)}{f^b_{i_k}} \tag{22}$$

where $f^b_{i_k}$ is the computing capability of UAV i_k. The service delay is the sum of computing delay and transmission delay:

$$T^e(k) = t^u(k) + t^b(k) \tag{23}$$

The reliability of the task k processed at UAV i_k is:

$$R^{edge}_k = R^c_{i_k,k} \cdot R^t_{m_k,i_k,k} = e^{-\alpha_{i_k} t^{co}_{i_k,k} - \beta_{m_k,i_k} t^u(k)} \tag{24}$$

where $t^{co}_{i_k,k} = c_k / f^b_{i_k}$ is the processing time.

The computation energy of UAV i_k is:

$$E^{com}_{i_k}(k) = \kappa_{i_k} c_k \tag{25}$$

where κ_{i_k} is the effective switched capacitance of UAV i_k.

C. Cloud Computing

Unlike UAVs, the EC has abundant computing resources and can allocate dedicated computing resources to each device for task processing directly. If task k is offloaded to EC, there are no computation queues and the computing delay is:

$$t^c(k) = \frac{c_k}{f^c_{m_k}} \tag{26}$$

where $f^c_{m_k}$ is the fixed computing capability that the cloud server allocates to device m_k.

Similar to edge computing, the transmission rate for UE m to the EC over the wireless link is:

$$Rate_m(k) = B^C log_2(1 + \frac{p_m(k) g_{m,N+1}(k)}{N_C}) \tag{27}$$

where B^C is channel bandwidth for UE-to-EC links, and N_C is the noise power. While employing a transmission model akin to that of UE-to-UAV, the UE-to-EC experiences distinct transmission delays. Due to its greater distance from UE m compared to UAV, cloud computing easily incurs longer transmission delays.

The service delay of cloud computing is:

$$T^c(k) = T^u(k) + t^c(k) \tag{28}$$

where $T^u(k) = u_k / Rate_{m_k}(k)$ is the transmission delay.

Similar to edge computing, the reliability of the task k processed at cloud server is

$$R^{cloud}_k = e^{-\alpha_{N+1} t^c(k) - \beta_{m_k,N+1} T^u(k)} \tag{29}$$

D. Problem Formulation

The completion time of task k is:

$$\begin{aligned} T(k) \quad = \quad & \mathbf{1}_{\{i_k=0\}} T^l(k) + \mathbf{1}_{\{i_k\neq 0 \& i_k \neq N+1\}} T^e(k) \\ & + \mathbf{1}_{\{i_k=N+1\}} T^c(k) \end{aligned} \tag{30}$$

The equation consists of three parts, which represent the delay of local computing, edge computing and cloud computing. For example, if $i_k = 0$, which means the task is processed locally, the remaining two parts except the first part are 0.

The task k should be completed before its deadline, which can be expressed as:

$$T(k) \leq d_k \tag{31}$$

Otherwise, the task is overdue and is automatically dropped from the edge node.

In UAV-assisted MEC systems, we jointly optimize the offloading decision i_k, UAVs position $\mathbf{w}_n(k)$ and transmit power of UEs $p_m(k)$ to maximize the success rate of tasks. A task k is successful if its completion time does not exceed its deadline d_k and can ensure error-free execution and transmission. The task success rate is defined as follows:

$$ST(k) = \frac{|K_{succ}(k)|}{|K_{succ}(k)| + |K_{drop}(k)| + |K_{fail}(k)|} \tag{32}$$

where $|K_{succ}(k)|$ represents the count of tasks successfully processed up to the arrival of task k, $|K_{drop}(k)|$ indicates the number of tasks that surpass their deadline and are consequently dropped from the edge network, and $|K_{fail}|$ denotes the tasks that fail during execution or transmission, which is related to the system reliability.

Considering the limited energy of UEs and UAVs, the energy consumption constraint of UAV n is:

$$E^{fly}_n(k) + \mathbf{1}_{\{i_k=n\}} E^{com}_n(k) \leq E^{max}_n(k), \forall n \in \{1, ..., N\} \tag{33}$$

The energy consumption constraint of UE m_k which generates task k:

$$\mathbf{1}_{\{i_k=0\}} \kappa_{m_k} c_k + \mathbf{1}_{\{i_k\neq 0\}} p_{m_k}(k) t^u(k) \leq E^{max}_{m_k}(k) \tag{34}$$

The left side of the formula is composed of two parts, the first part represents the computation energy consumption of the UAV, and the second part represents the transmission energy consumption of the UAV.

The optimization problem is formulated as follows:

$$\begin{aligned} & \max \lim_{k\to\infty} ST(k) \\ s.t. \quad & 0 \leq p_m(t) \leq P^{UE}_{max}, \quad \forall m \in \mathcal{M} && \text{(35a)} \\ & i_k \in \{0, 1, ..., N+1\} && \text{(35b)} \\ & (1) - (3), (33), (34) && \text{(35c)} \end{aligned}$$

where the optimization goal is to maximize the long-term task success rate of the system. Constraint (35a) denotes the limited transmit power of UE, constraint (35b) indicates the value of offloading decision variable, Eq. (1)-(3) describe the position constraints of UAVs, and Eq.(33),(34) are the limited energy consumption of UAVs and UEs.

E. MDP Formulation

In this optimization problem, the next state of the system is only related to the current state and action, which means the problem can be formulated as a MDP. Considering that the system evolution is described by the arrival of tasks, we also describe state transitions through increments in computational tasks.

1) State Space: The current workload of UE m_k is a crucial factor influencing the decision. We define the current workload of UE m_k as the waiting delay before task k:

$$t^d(k) = \frac{[q_{m_k}(k-1) - (t_k - t_{k-1}) f^d_{m_k}]^+}{f^d_{m_k}} \tag{36}$$

where the numerator represents the backlog of computational tasks on UE m_k upon task k arrival, and the denominator denotes the computing capability of UE m_k.

Similar to UEs, the current workload of each UAV n is also a critical factor influencing the decision. We also define it using the computing waiting delay:

$$T^d_n(k) = \frac{[Q_n(k-1) - (t_k - t_{k-1}) f^b_n]^+}{f^b_n} \tag{37}$$

Furthermore, the system state should include the transmission delay and certain characteristics of the task. The transmission delay of task k to other nodes (UAVs or EC) can be represented as a vector:

$$\mathbf{T^u}(\mathbf{k}) = [T^u_1(k), T^u_2(k), ..., T^u_N(k), T^u_{N+1}(k)] \tag{38}$$

To be specific, the system state when task k arrives is defined as:

$$s(k) = (t^d(k), f^d_{m_k}, \mathbf{T^d}(\mathbf{k}), \mathbf{T^u}(\mathbf{k}), \mathbf{f^b}, f^c_{m_k}, c_k, d_k) \tag{39}$$

where $\mathbf{T^d}(\mathbf{k}) = [T^d_1(k), T^d_2(k), ..., T^d_N(k)]$ is the vector composed of the current workload of all UAVs and $\mathbf{f^b} = [f^b_1, f^b_2, ..., f^b_N]$ is the vector composed of the computing capability of all UAVs. The system space is composed of eight variables. The dimension of $\mathbf{T^d}(\mathbf{k})$ is N, the dimension of $\mathbf{T^u}(\mathbf{k})$ is $N+1$, the dimension of $\mathbf{f^b}$ is N, and the dimension of the remaining five variables is 1, so the dimension of the system space is $dim = 3N + 6$.

2) Action Space: In this problem, we jointly optimize the offloading decision, mobility of UAVs and transmit power of UE. The action is defined as:

$$a(k) = \{i_k, \mathbf{w}(k), p_{i_k}(k)\} \tag{40}$$

The action space is composed of three variables. As $\mathbf{w}(k)$ represents the 3D position of the UAVs, and its dimension is $3N$, the dimension of action $a(t)$ is $3N + 2$. It is worth mentioning that this is a discrete-continuous hybrid action space, where i_k is a discrete variable and $\mathbf{w}(k)$ and $p_{i_k}(k)$ are continuous variables.

3) Reward Function: Upon successful processing of task k, a feedback of +1 is received; otherwise, if task k is dropped or fails, a feedback of -1 is received later. All nodes collectively share a team reward, defined as the sum of the feedbacks across all nodes.

V. Algorithm Design Based on DRL

Traditional DRL algorithms which convert hybrid action space into either a discrete or a continuous action space directly may reduce the model performance due to scalability issue and increased approximation. In this section, we design a novel DRL algorithm based on a hybrid action representation [28] to address these challenges.

A. Dependence-aware Latent Space

There exists a discrete variable i_k and continuous variables ($\{\mathbf{w}(k), p_{i_k}(k)\}$). Our hybrid action representation transforms the formulated MDP into a continuous policy learning problem that captures interdependence between heterogeneous components. To simplify notation, we use v to uniformly denote continuous actions and remove the subscript k (i.e., action $a = (i, v)$) for clarity in the algorithm. Discrete and continuous variables in the action space are intertwined and jointly affect the environment. Only representing discrete variables may inadequately account for the relationship between discrete and continuous variables. In contrast, our representation method simultaneously trains the entire action space.

There are $N + 2$ computation offloading locations for each task. Initially, we define an embedding table $G_\omega \in \mathbb{R}^{(N+2)\times l_1}$ with trainable parameters ω to represent the $N + 2$ discrete actions. Each row $g_{\omega,i} = G_\omega(i)$ in the table is a l_1-dimensional continuous vector corresponding to discrete action i. This embedding table is not predefined but rather learned during training, alongside the continuous variables. The specific loss function will be detailed subsequently.

Next, we employ a conditional Variational Auto-Encoder (VAE) [29] to design a l_2-dimensional space for continuous variables. In mathematical terms, for a hybrid action $a = (i, v)$ and a state s, the encoder $q_\phi(z|v, s, g_{\omega,i})$, parameterized by ϕ, maps v to the latent variable $z \in \mathbb{R}^{l_2}$ conditioned on s and $g_{\omega,i}$. A Gaussian latent distribution $\Gamma(\mu_q, \sigma_q)$ is used to model the encoder $q_\phi(z|v, s, g_{\omega,i})$, where μ_q and σ_q denote the mean and standard deviation outputs, respectively.

Under the same framework, the decoder $q_\psi(\tilde{v}|z, s, g_{\omega,i})$, parameterized by ψ , reconstructs the continuous variable $\tilde{v}$ from z. Given any sampled $z \sim \Gamma(\mu_q, \sigma_q)$, the decoder performs deterministic decoding, expressed as $\tilde{v} = q_\psi(z, s, g_{\omega,i})$. Additionally, by conducting a nearest-neighbor lookup in the embedding table associated with $g_{\omega,i}$, we retrieve the discrete action i.

Through the encoder, we create a hybrid action representation space ($\in \mathbb{R}^{l_1+l_2}$) tailored for hybrid actions. Furthermore, we are able to decode latent variables $g \in \mathbb{R}^{l_1}$ and $z \in \mathbb{R}^{l_2}$ back into their hybrid action (i, v) equivalents as dictated by the decoder. Below is a formal summary of the encoding and decoding processes:

Encoding:

$$g_{\omega,i} = G_\omega(i), \qquad z \sim q_\phi(v, s, g_{\omega,i}) \tag{41}$$

Decoding:

$$i = argmin_{i' \in \mathcal{I}} ||g_{\omega,i'} - g||_2, \qquad v = q_\psi(z, s, g_{\omega,i}) \tag{42}$$

By utilizing experiences in buffer $\mathcal{D}$, the loss function which is the sum of squared L_2-norm reconstruction error and Kullback-Leibler divergence:

$$L_V(\psi,\phi,\omega) = \mathbb{E}[||v-\tilde{v}||_2^2 + L(q_\phi(\cdot|v,s,g_{\omega,i})|\Gamma(0,I))] \quad (43)$$

Considering the diverse impacts of hybrid actions on the environment, we employ a cascaded architecture situated following the conditional VAE decoder. For any experience instance (s,i,v,s'), our decoder featuring the cascaded configuration can produce predictions in the following manner:

$$\bar{\delta}_{s,s'} = q_\psi(z,s,g_{\omega,i}), \quad for \quad z,s,g_{\omega,i} \quad (44)$$

Then the L_2-norm square prediction error is:

$$L_D(\psi,\phi,\omega) = \mathbb{E}[||\bar{\delta}_{s,s'} - \delta_{s,s'}||_2^2] \quad (45)$$

where $\delta_{s,s'} = s' - s$ denotes the residual state.

We jointly train q_ϕ, q_ψ and G_ω by minimizing the following training loss:

$$L_H(\psi,\phi,\omega) = L_V(\psi,\phi,\omega) + \alpha L_D(\psi,\phi,\omega) \quad (46)$$

where α is a weight parameter dependent on the significance of predictive representation loss in dynamics. The architectures of the encoder and decoder networks are detailed in Table I.

B. Problem Solving using DRL

In this section, we integrate the representation space with TD3 algorithm [30] to address the optimization problem.

TD3, a deterministic strategy DRL algorithm, is well-suited for navigating high-dimensional continuous action spaces. It consists of two primary networks. On one hand, the actor network maps diverse states to actions. On the other hand, the critic network evaluates the potential scores for various actions under varying state, thereby influencing the action values.

We combine the representation space approach with the TD3 algorithm to propose a novel DRL method tailored for discrete-continuous hybrid action spaces problem. In TD3, the actor and critic functions are realized using distinct neural networks, detailed in Tab. II. The actor network takes the system state s as input and outputs a latent action vector $g,z = \pi_\zeta(s)$, where $g \in \mathbb{R}^{l_1}, z \in \mathbb{R}^{l_2}$. Subsequently, a decoder transforms latent vector (g,z) into the hybrid action (i,v). The double critic networks $Q_{\theta_1}, Q_{\theta_2}$ evaluate the hybrid-action value function Q^{π_ζ} using (i,v) as input. During training with experiences (s,i,v,r,s') stored in buffer $\mathcal{D}$, we update the critic networks using the loss function:

$$L_{CDQ}(\theta_j) = \mathbb{E}[(o - Q_{\theta_j}(s,g,z))^2], \quad for \quad j=1,2 \quad (47)$$

where $o = r + \gamma min Q_{\bar{\theta}_j}(s', \pi_{\bar{\zeta}}(s'))$ and $\bar{\theta}_j, \bar{\zeta}$ are the target network parameters. We use deterministic policy gradient to update the actor network:

$$\nabla_\zeta J(\zeta) = \mathbb{E}[\nabla_{\pi_\zeta(s)} Q_{\theta_1}(s,\pi_\zeta(s)) \nabla_\zeta \pi_\zeta(s)] \quad (48)$$

We propose the Task-driven Reliability-aware Offloading (TRO) algorithm to address this problem. The detailed TRO algorithm is presented in Algorithm 1. Initially, network parameters and embedding tables are randomly initialized, with the system state $s(1)$ set to the UAV's starting positions. Training progresses through two main stages: warm-up and learning. During warm-up (lines 3-5), the encoder and decoder are pre-trained using experiences from the replay buffer $\mathcal{D}$. In the learning stage, the actor generates a latent action g,z based on the current state s. Subsequently, the decoder transforms g,z into i,v, yielding reward r and new state s'. This experience (s,i,v,g,z,r,s') is stored in $\mathcal{D}$. To mitigate input sample correlations, mini-batch experiences are randomly sampled from $\mathcal{D}$. The loss function is computed based on critic network evaluations, and parameters of networks are updated (lines 15-16). Concurrently, the encoder and decoder are updated during training to adapt to changing data distributions (lines 17-19). The actor network can operate independently of the critic network (lines 7-12) once fully trained.

Algorithm 1 TRO training Algorithm

1: Initialize embedding table, conditional VAE, actor, critic with random parameters;
2: Initialize state information s_1;
3: **while** not reach maximum warm-up training times **do**
4: Update ω,ϕ,ψ using samples in $\mathcal{D}$ by Eq.(46);
5: **end while**
6: **while** not reach maximum total environment steps **do**
7: /* select latent actions by actor network */
8: $g,z = \pi_\zeta(s) + \epsilon_g$ with $\epsilon_g \sim \Gamma(0,\sigma)$;
9: /* decode into original hybrid actions */
10: Decode $i = f_D(g), v = q_\psi(z,s,g)$ by decoder;
11: Execute (i,v), get reward r and new state s';
12: Store (s,i,v,g,z,r,s') in replay buffer $\mathcal{D}$;
13: /* evaluate hybrid actions by critic network */
14: Sample a mini-batch experience from $\mathcal{D}$;
15: Update $Q_{\theta_1}, Q_{\theta_2}$ according to the loss function Eq.(47);
16: Update π_ζ with policy gradient according to Eq.(48);
17: **while** not reach representation training times **do**
18: Update ω,ϕ,ψ using samples in $\mathcal{D}$ by Eq.(46);
19: **end while**
20: **end while**

C. Complexity Analysis

The complexity of our proposed algorithm consists primarily of two components: the complexity of encoder and decoder, and the complexity associated with training the actor and critic networks. According to Sipper [31], or a fully connected neural network with fixed numbers of hidden layers and neurons, the computational complexity scales proportionally with the product of input size and output size. In our algorithm, the four networks all have fixed numbers of hidden layers and neurons. Therefore, we first calculate the computational complexity of the four neural networks by the product of input size and output size. Then, the overall computational complexity of the algorithm is the sum of the individual computational complexities of the four networks.

In the encoding and decoding of hybrid actions, the input size of the encoder is $dim + l_1 + 1 = 3N + 7 + l_1$. The output size of the encoder is l_2, resulting in a complexity of $\mathcal{O}((N+l_1)l_2)$ for the encoder. The input size of the decoder

TABLE I
NETWORK STRUCTURES OF ENCODER AND DECODER

Model Component	Layer	Structure	Layer	Structure
Discrete Action Embedding Table G_ω	Parameterized Table	$(\mathbb{R}^{N+2}, \mathbb{R}^{l_1})$		
Conditional Encoder Network $q_\phi(z\|v, s_k, g_{\omega,i})$	Fully Connected(encoding) Fully Connected(condition) Element-wise Product Fully Connected Activation	$(1, 128)$ $(dim + \mathbb{R}^{l_1}$,128) ReLU · ReLU (128,128) ReLU	Fully Connected(mean) Activation Fully Connected(log_std) Activation	(128,$\mathbb{R}^{l_2}$) None (128,$\mathbb{R}^{l_2}$) None
Conditional Decoder & Prediction Network $q_\psi(\tilde{v}\|z, s_k, g_{\omega,i})$	Fully Connected(latent) Fully Connected(condition) Element-wise Product Fully Connected Activation Fully Connected(reconstruction)	$(\mathbb{R}^{l_2}, 128)$ $(dim + \mathbb{R}^{l_1}$,128) ReLU · ReLU (128,128) ReLU (128,1)	Activation Fully Connected Activation Fully Connected(prediction) Activation	None (128,128) ReLU (128,dim) None

TABLE II
NETWORK STRUCTURES OF TD3

Model Component	Layer	Structure	Layer	Structure
Actor Network π_ς	Fully Connected Activation Fully Connected	$(dim, 128)$ ReLU (128,128)	Activation Fully Connected Activation	ReLU (128,$\mathbb{R}^{l_1+l_2}$) Tanh
Critic Network Q_{θ_j}	Fully Connected Activation Fully Connected	$(dim + 2, 128)$ ReLU (128,128)	Activation Fully Connected Activation	ReLU (128,1) None

is $dim + l_1 + l_2 = 3N + 6 + l_1 + l_2$. The output size of the decoder is $dim + 1 = 3N + 7$, leading to a complexity of $\mathcal{O}((N + l_1 + l_2)N)$ for the decoder.

In training the actor and critic networks, the input size of the actor network is the dimension of system space $dim = 3N+6$, and the output size is $l_1 + l_2$, resulting in a complexity of $\mathcal{O}((l_1+l_2)N)$ for the actor. The input size of the critic network is $dim + 2$, and the output size is 1, yielding a complexity of $\mathcal{O}(N)$ for the critic.

Thus, the complexity of our algorithm per iteration is $\mathcal{O}((N+l_1)l_2)+\mathcal{O}((N+l_1+l_2)N)+\mathcal{O}((l_1+l_2)N)+\mathcal{O}(N) = \mathcal{O}(N^2 + Nl_1 + Nl_2 + l_1l_2)$.

We also analyse the memory footprint of our algorithm. The memory footprint of our algorithm is primarily influenced by the space complexity of neural networks, largely determined by the number of model parameters. For a layer of fully connected neural network, the parameter number is the product of input size and output size. For a network, the parameter number is the sum of the parameters of each layer. Therefore, we calculate the parameters of each layer separately. Here is a breakdown of the space complexity for each component.

Encoder Network: The number of parameters is $\mathcal{O}(1\times128+(dim + l_1) \times 128 + 128 \times 128 + 128 \times l_2 + 128 \times l_2) = \mathcal{O}(N + l_1 + l_2)$.

Decoder Network: The number of parameters is $\mathcal{O}(l_2 \times 128 + (dim + l_1) \times 128 + 128 \times 128 + 128 \times 1 + 128 \times 128 + 128 \times dim) = \mathcal{O}(N + l_1 + l_2)$.

Actor Network: There are three full layers of connected neural network in the actor network and its complexity is $\mathcal{O}(dim \times 128 + 128 \times 128 + 128 \times (l_1 + l_2)) = \mathcal{O}(N + l_1 + l_2)$.

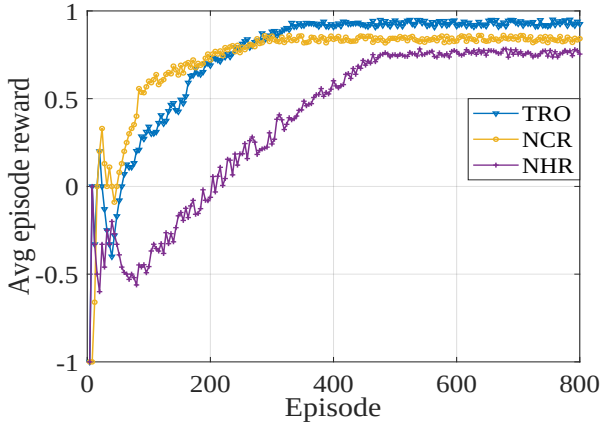


Fig. 2. Convergence of algorithms

Critic Network: The complexity based on the number of parameters is $\mathcal{O}(dim \times 128 + 128 \times 128 + 128 \times 1) = \mathcal{O}(N)$.

Therefore, the space complexity of the neural networks in our algorithm sums up to $\mathcal{O}(N + l_1 + l_2)$. Additionally, the algorithm's memory footprint includes other factors such as the replay buffer and distributed training setup, which vary based on the specific implementation details.

VI. PERFORMANCE EVALUATION

In this section, first we outline the setup of our simulation experiments and present four baseline methods for comparative analysis. Next simulations are conducted to assess the efficacy of our algorithm. Then, we build a Kubernetes-based testbed, where nodes can communicate through a TCP/IP

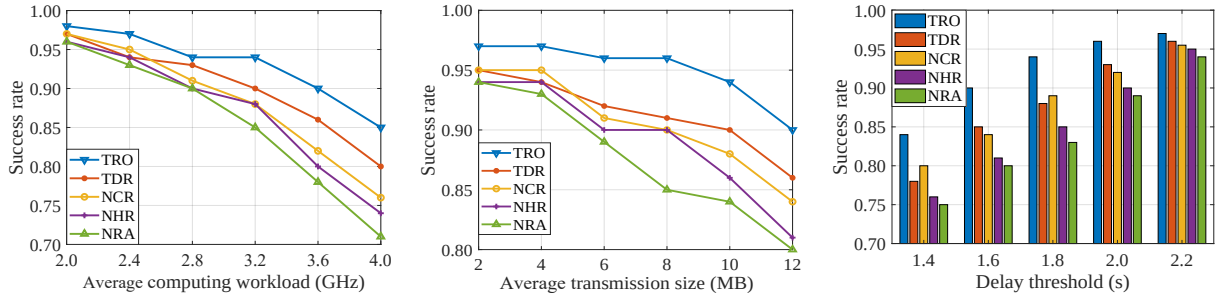


Fig. 3. Successful rate vs. (a) average computing workload of tasks (b) average transmission size of tasks (c) delay threshold of tasks

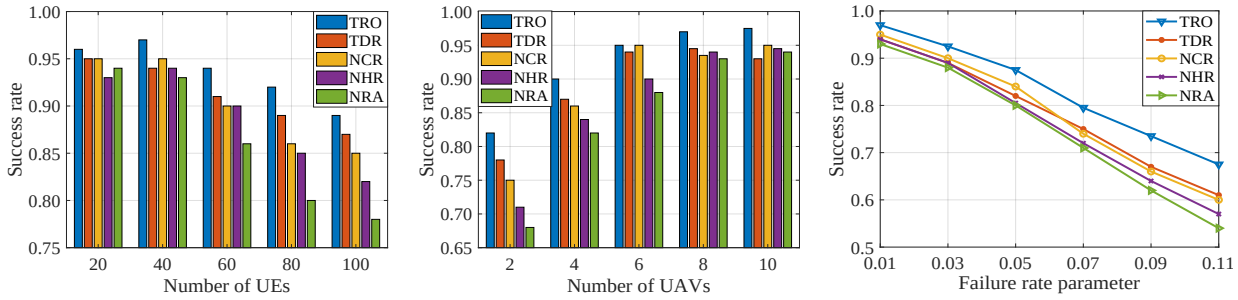


Fig. 4. Successful rate vs. (a) the number of UAVs (b) the number of UEs (c) failure rate parameter

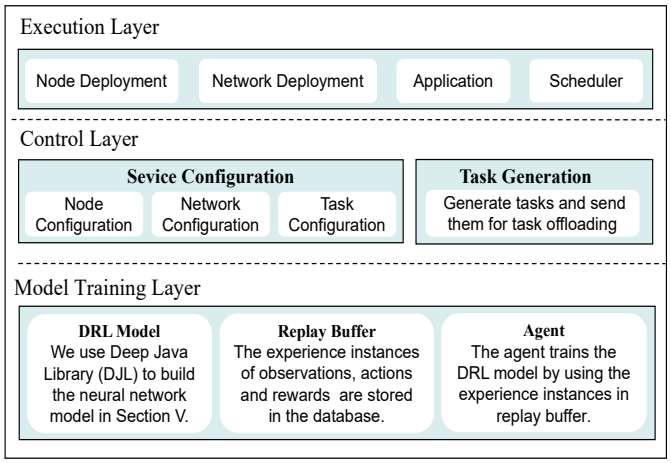


Fig. 5. Experiment Prototype

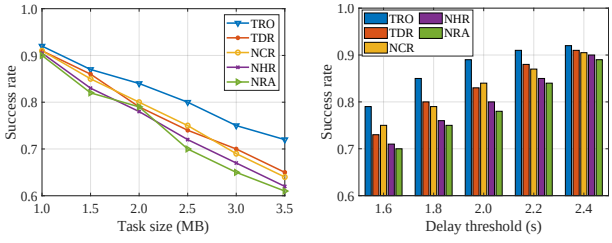


Fig. 6. Experiment in Kubernetes-based Testbed

network, in order to evaluate the performance of algorithms in more realistic scenarios.

A. Simulation Setup

In this section, we perform simulation experiments on the proposed TRO algorithm using Python 3.9 and PyTorch 2.2. For the specified region of the UAV-supported MEC network offering varied services, we analyze a square area measuring 400 meters on each side, i.e., $x_{\max} = y_{\max} = 400m$. For each UAVs, the computing capability f_n is 10 Gigacycles. We set the maximum horizontal speed L^h_{max} to 25m/s, maximum vertical speed L^v_{max} to 10m/s, effective switched capacitance κ to 10^{-28}, and noise power N_A,N_C to -100dBm. Constant values and excessive path loss a, b, η_{LoS}, and η_{NLoS} are set to 9.61, 0.16, 1, and 20, respectively [14]. The computing capability of UEs f_m is uniformly distributed [2,4] Gigacycles, the channel bandwidth for UE-to-UAV links B^A is uniformly distributed [10,20] MB, and the channel bandwidth for UE-to-EC links B^C is uniformly distributed [2,5] MB. As for the neural network, the batch size of training is 64, the discount factor is 0.99, and the optimizer is Adam. We compare our TRO algorithm with the following four alternative solutions.

- **Time-driven and reliability-aware algorithm (TDR) [19]**: TDR is based on a novel DRL solution. It discretizes the timeline and is driven by time slots. TDR incorporates task reliability considerations during both execution and transmission phases, but does not consider the mobility of UAVs.
- **No Continuous variables representation algorithm (NCR)**: To verify the effectiveness of hybrid action representation of our TRO algorithm, we design two algorithms to perform ablation experiments, and NCR is one of them. NCR does not consider the correlation between variables, and only represents discrete variables rather than continuous variables.
- **No Hybrid action representation algorithm (NHR)**: The NHR is the other algorithm to perform ablation experiments. It does not have any representation of variables, and only uses a simple rounding approach to

change the continuous values into discrete values.

- **No consideration of reliability algorithm (NRA) [32]**: The optimization objective of NRA algorithm was to minimize the average task delay rather than the task success rate. It does not consider the reliability of execution or transmission.

B. Simulation Results

Fig.2 illustrates the ablation experiment with respect to the convergence of algorithms. The NHR algorithm only uses a simple rounding approach to change the continuous values into discrete values, which determines large fluctuations during model training and increases the difficulty of model training. So the convergence of NHR algorithm is the worst. The model convergence rate of NCR which only represents discrete variables is the fastest. The reason is that it does not have the fluctuation caused by rounding, and has a simple network structure. Our TRO algorithm needs to represent discrete variables and continuous variables, which involves a more complex neural network structure than the NCR algorithm. So the convergence rate of TPO is slightly worse than that of NCR. Overall, it can be seen from the figure that our algorithm still has a very good convergence.

Fig.3 shows the effect of task-related attributes (i.e. computing workload c_k, transmission size t_k and delay threshold d_k) on the task success rate of the compared algorithms. The effect of average computing workload is shown in Fig.3(a). The average computing workload becomes larger, but the amount of total computing resources is limited, which means that it often takes more time to complete the task. Therefore, the task success rate of all five algorithms decreases as the average computing workload of tasks increases. However, in general, our proposed TRO algorithm outperforms the other four algorithms. The performance of TDR and NCR is similar to each other, and slightly worse than that of our TRO algorithm. The reason is that TDR is timeslot-driven which affects its performance, while NCR ignores the relationship between discrete variables and continuous variables. The NHR algorithm performs worse because it do not represent the variables. The NRA algorithm has the worst performance because it does not consider the reliability of system and the optimization goal is not the success rate.

Fig.3(b) shows the effect of average transmission size. Similar to average computing workload, the success rate of all five algorithms decreases as average transmission size increases. This is because the increased transmission size means higher transmission delay and lower transmission reliability. The effect of delay threshold on success rate is opposite to that of average computing workload and average transmission size, which is shown as Fig.3(c). As the increase of delay threshold, the number of tasks that are dropped due to timeouts decreases dramatically, so the success rate of task increases. However, the success rate increases slowly and the success rate is always not equal to 1, because there will be tasks that encountered failure during execution or transmission, which is related to the system reliability.

We analyze the effect of some other factors on the success rate in Fig.4. Fig.4(a) shows that the success rate decreases as the number of users increases. More users means more computation workload on the system. Due to the limited computation resource, the task delay will increases, thus causing more tasks to exceed the delay threshold and fail. By the way, the success rate does not decrease significantly in the early stage of UE growth, because the computing capacity is relatively sufficient at this time, and the increase of UEs does not bring obvious computation pressure. The increase in the number of UAVs is the opposite of UEs, which is shown in Fig.4(b). An increase in the number of UAVs represents an increase in computation resources. The computation latency is reduced and more tasks can be completed before their deadlines. But, the success rate also is not equal to 1 due to the reliability of system. The fault injection is performed by varying the failure rate parameter. The larger the failure rate parameter, the more frequent the faults occur. From Fig.4(c), we can find that the effect of the failure rate parameter of UAVs on task success rate is significant. As the failure rate parameter becomes larger, the frequency of fault injection is higher and the reliability of UAVs is worse, making more tasks encounter failure during execution. As a result, the task success rate decreases.

C. Experiment in the Kubernetes-based Testbed

In the simulation experiment, each node is just an instance in the program and the communication between nodes is simulated, as there is no actual communication based on the TCP/IP protocol. To validate the practicality and applicability of our algorithm, we have built a testbed based on Kubernetes. The simulation experiments and experiments in Kubernetes-based testbed are complementary to each other.

The architecture of our Kubernetes-based testbed is shown in Fig. 5. There are three layers: model training layer, control layer and execution layer. The bottom layer is model training layer, which is responsible for the training of model. There are three parts, which include DRL model, replay buffer and agent. We implemented the algorithm based on Deep Java Library (DJL), which is a deep learning framework for Java. The middle layer is the control layer, which runs as a Pod in the Kubernetes. The control layer is responsible for configuring the network service environment and generation of tasks. The top layer is the execution layer, which sets up the nodes and makes task offloading decisions within the application.

Fig.6 shows the experiment results in Kubernetes-based testbed. In the experiment, the computing workload of task is proportional to the data transmission size. For example, the larger the amount of video transmission data, the larger the amount of computing workload brought by its encoding and decoding. In order to reflect this property and avoid confusion with transmission size in simulation experiments, we use task size here. Fig.6(a) shows that the impact of task size on the success rate is quite significant. Increasing the task size will drastically decrease the success rate. The reason is that an increase in task size means that both transmission size and computing workload will increase simultaneously. We also made experiments in delay threshold of tasks, as shown in Fig.6(b). Similar to simulation results, the increase of delay threshold will improve the success rate.

VII. Conclusions

This paper investigates the task-driven task offloading problem in a multiple UAV-assisted MEC system. Considering the system reliability, we formulate a joint optimization problem for UAV trajectories, task offloading, and communication resource management, whose goal is to maximize long-term average value of task success rate. Then, we transform the optimization problem into a MDP by defining state, action and reward. Due to the discrete-continuous hybrid action space of problem, conventional DRL algorithms are not applicable. We propose a dependence-aware latent space representation algorithm to represent discrete-continuous hybrid action. Based on this, we design a novel DRL algorithm to solve the problem. Extensive simulation experimental results shows that our algorithm can achieve better performance compared to four alternative baseline approaches. Furthermore, we build a Kubernetes-based Testbed to validate the practicality and applicability of our algorithm.

The decentralized system also has its advantages, especially in terms of flexibility and robustness. Future work will carry out in-depth research on completely decentralized systems. As real-life experiments are the best validation of any algorithm, in the future, we will strive to perform real experimental testing.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NSFC) via grant 62401304, 62225105; Shandong Provincial Natural Science Foundation under Grant. ZR2022QF040; the Young Talent of Lifting engineering for Science and Technology in Shandong, China via SDAST2025QTA077, the QLU Talent Research Project under grant 2023RCKY138, and the Taishan Scholar Program of Shandong Province in China under Grant No.TSQN202312230.

References

[1] Y. Mao et al., "A Survey on Mobile Edge Computing: The Communication Perspective," *EEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, 2017.

[2] W. Lee and T. Kim, "Multiagent Reinforcement Learning in Controlling Offloading Ratio and Trajectory for Multi-UAV Mobile-Edge Computing," *IEEE Internet of Things Journal*, vol. 11, no. 2, pp. 3417-3429, 2024.

[3] Q. Luo, T. H. Luan, W. Shi and P. Fan, "Deep Reinforcement Learning Based Computation Offloading and Trajectory Planning for Multi-UAV Cooperative Target Search," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 2, pp. 504-520, 2023.

[4] Z. Kuang, Y. Pan, F. Yang and Y. Zhang, "Joint Task Offloading Scheduling and Resource Allocation in Air–Ground Cooperation UAV-Enabled Mobile Edge Computing," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 4, pp. 5796-5807, 2024.

[5] F. Shirin Abkenar et al., "A Survey on Mobility of Edge Computing Networks in IoT: State-of-the-Art, Architectures, and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2329-2365, 2022.

[6] R. Khalid, Z. Shah, M. Naeem, A. Ali, A. Al-Fuqaha and W. Ejaz, "Computational Efficiency Maximization for UAV-Assisted MEC Networks With Energy Harvesting in Disaster Scenarios," *IEEE Internet of Things Journal*, vol. 11, no. 5, pp. 9004-9018, 2024.

[7] Z. Wei, B. Zhao and J. Su, "Event-Driven Computation Offloading in IoT With Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6847-6860, 2022.

[8] J. Zheng et al., "Dynamic Computation Offloading for Mobile Cloud Computing: A Stochastic Game-Theoretic Approach," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 771-786, 2019.

[9] H. Hao, C. Xu, W. Zhang, S. Yang and G. -M. Muntean, "Task-Driven Priority-Aware Computation Offloading Using Deep Reinforcement Learning," *IEEE Transactions on Wireless Communications* , early access, 2025.

[10] Z. Ning et al., "5G-Enabled UAV-to-Community Offloading: Joint Trajectory Design and Task Scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 11, pp. 3306-3320, 2021.

[11] T. Bai, J. Wang, Y. Ren and L. Hanzo, "Energy-Efficient Computation Offloading for Secure UAV-Edge-Computing Systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6074-6087, 2019.

[12] Z. Yu, Y. Gong, S. Gong and Y. Guo, "Joint Task Offloading and Resource Allocation in UAV-Enabled Mobile Edge Computing," *IEEE Internet of Things Journal*, vol.7, no. 4, pp. 3147-3159, 2020.

[13] B. Xu, Z. Kuang, J. Gao, L. Zhao and C. Wu, "Joint Offloading Decision and Trajectory Design for UAV-Enabled Edge Computing With Task Dependency," *IEEE Transactions on Wireless Communications*, vol. 22, no. 8, pp. 5043-5055, 2023.

[14] H. Guo et al., "Multi-UAV Cooperative Task Offloading and Resource Allocation in 5G Advanced and Beyond," *IEEE Transactions on Wireless Communications*, vol. 23, no. 1, pp. 347-359, 2024.

[15] S. Tong, Y. Liu, J. Mišić, X. Chang, Z. Zhang and C. Wang, "Joint Task Offloading and Resource Allocation for Fog-Based Intelligent Transportation Systems: A UAV-Enabled Multi-Hop Collaboration Paradigm," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 11, pp. 12933-12948, 2023.

[16] Z. Bai, Y. Lin, Y. Cao and W. Wang, "Delay-Aware Cooperative Task Offloading for Multi-UAV Enabled Edge-Cloud Computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1034-1049, 2024.

[17] X. Dai, Z. Xiao, H. Jiang and J. C. S. Lui, "UAV-Assisted Task Offloading in Vehicular Edge Computing Networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2520-2534, 2024.

[18] W. Lee and T. Kim, "Multiagent Reinforcement Learning in Controlling Offloading Ratio and Trajectory for Multi-UAV Mobile-Edge Computing," *IEEE Internet of Things Journal*, vol. 11, no. 2, pp. 3417-3429, 2024.

[19] H. Lin, L. Yang, H. Guo and J. Cao, "Decentralized Task Offloading in Edge Computing: An Offline-to-Online Reinforcement Learning Approach," *IEEE Transactions on Computers*, vol. 73, no. 6, pp. 1603-1615, 2024.

[20] H. Hao, C. Xu, W. Zhang, S. Yang and G. -M. Muntean, "Joint Task Offloading, Resource Allocation, and Trajectory Design for Multi-UAV Cooperative Edge Computing with Task Priority," *IEEE Transactions on Mobile Computing*, vol. 23, no. 9, pp. 8649-8663, 2024.

[21] Z. Ning, Y. Yang, X. Wang, Q. Song, L. Guo and A. Jamalipour, "Multi-Agent Deep Reinforcement Learning Based UAV Trajectory Optimization for Differentiated Services," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 5818-5834, 2024.

[22] F. Song et al., "Evolutionary Multi-Objective Reinforcement Learning Based Trajectory Control and Task Offloading in UAV-Assisted Mobile Edge Computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 7387-7405, 2023.

[23] L. Wang et al., "Multi-Agent Deep Reinforcement Learning-Based Trajectory Planning for Multi-UAV Assisted Mobile Edge Computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 1, pp. 73-84, 2021.

[24] Z. Liu, K. Li, L. Wu, Z. Wang, Y. Yang. "CATS: Cost Aware Task Scheduling in Multi-Tier Computing Networks," *Journal of Computer Research and Development*, vol. 57, no. 9, pp. 1810-1822, 2020.

[25] X. Zhang et al., "Energy-Efficient Multi-UAV-Enabled Multiaccess Edge Computing Incorporating NOMA," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5613-5627, 2020.

[26] J. Liu et al., "Reliability-Enhanced Task Offloading in Mobile Edge Computing Environments," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 10382-10396, 2022.

[27] J. Jia, L. Yang and J. Cao, "Reliability-aware Dynamic Service Chain Scheduling in 5G Networks based on Reinforcement Learning," *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, Vancouver, BC, Canada, 2021, pp. 1-10.

[28] B. Li, H. Tang, Y. Zheng, et al., "HyAR: Addressing Discrete-Continuous Action Reinforcement Learning via Hybrid Action Representation," *International Conference on Learning Representations (ICLR)*, 2022.

[29] D. P. Kingma and M. Welling. "Auto-encoding Variational Bayes," *International Conference on Learning Representations (ICLR)*, 2014.

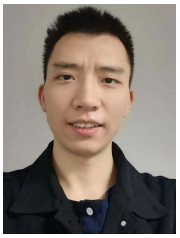
[30] S. Fujimoto, H. v. Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods," *International Conference on Machine Learning (ICML)*, pp. 1582–1591, 2018.
[31] M. Sipper, "A serial complexity measure of neural networks," *IEEE International Conference on Neural Networks*, pp. 962-966 vol.2, 1993.
[32] M. D. Nguyen, L. B. Le and A. Girard, "Integrated Computation Offloading, UAV Trajectory Control, Edge-Cloud and Radio Resource Allocation in SAGIN," *IEEE Transactions on Cloud Computing*, vol. 12, no. 1, pp. 100-115, 2024.

BIOGRAPHIES

Hao Hao received the Ph. D degree in computer science and technology from Beijing University of Posts and Telecommunications, Beijing, China,in 2021. He is currently a lecturer with the Shandong Computer Science Center (National Supercomputing Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences). His research interests include MEC and content caching over the wireless network, multimedia communications.

Changqiao Xu (SM'15) received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences (ISCAS) in Jan. 2009. He was a researcher at Athlone Institute of Technology and joint PhD at Dublin City University, Ireland during 2007-2009. Currently, he is a Full Professor with the State Key Laboratory of Networking and Switching Technology, and Director of the Next Generation Internet Technology Research Center at Beijing University of Posts and Telecommunications (BUPT). His research interests include Future Internet Technology, Mobile Networking, Multimedia Communications. He has published over 200 technical papers in prestigious international journals and conferences, including IEEE Comm. Surveys & Tutorials, IEEE Wireless Comm., IEEE Comm. Magazine, IEEE/ACM ToN, etc. He has served many international conferences and workshops as Co-Chair or Technical Program Committee member. He is currently serving as the Editor-in-Chief of Transactions on Emerging Telecommunications Technologies (Wiley).

Wei Zhang received the B.E. degree from Zhejiang University in 2004, the M.S. degree from Liaoning University in 2008, and the Ph.D. degree from Shandong University of Science and Technology in 2018. He is currently a Professor with the Shandong Computer Science Center (National Supercomputing Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences). His research interests include future generation network architectures, edge computing and edge intelligence.



Xingyan Chen received the Ph. D degree in computer technology from Beijing University of Posts and Telecommunications (BUPT), in 2021. He is currently a lecturer with the School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu. He has published papers in well-archived international journals and proceedings, such as the IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and IEEE INFOCOM etc. His research interests include Multimedia Communications, Multi-agent Reinforcement Learning and Stochastic Optimization.

Shujie Yang received the Ph.D. degree from the Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing, China, in 2017, where he is currently a Lecturer with the State Key Laboratory of Networking and Switching Technology. His major research interests are in the areas of wireless communications and wireless networking.

Gabriel-Miro Muntean (F'23) is Professor with the School of Electronic Engineering, Dublin City University (DCU), Ireland, and co-Director of the DCU Performance Engineering Laboratory. He has published over 500 papers in top-level international journals and conferences, authored 4 books and 29 book chapters, and edited 6 additional books. He has supervised to completion 28 PhD students and has mentored 20 post-doctoral researchers and fellows. His research interests include quality, performance, and energy saving issues related to rich media content delivery, technology enhanced learning, and other data communications over heterogeneous networks. He is an Associate Editor of the IEEE TRANSACTIONS ON BROADCASTING, Multimedia Communications Area Editor of the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, and chair and reviewer for important international journals, conferences, and funding agencies. He was Project Coordinator and DCU team leader for the EU projects NEWTON, TRACTION and HEAT.