Computation Offloading Optimization for UAV-Based Cloud-Edge Collaborative Task Scheduling Strategy

Haosheng Chen, Haixia Cui, *Senior Member, IEEE,* Jiahuan Wang, Peng Cao, Yejun He, *Senior Member, IEEE*, and Mohsen Guizani, *Fellow, IEEE*

***Abstract*—Unmanned aerial vehicles (UAVs) with communication and computing capabilities are essential for supporting mobile edge computing (MEC). They can be deployed flexibly to provide computing services for remote mountainous areas or disaster-stricken areas with damaged communication facilities, thereby enhancing communication quality and coverage area. Considering the 5G ultra-reliable and low-latency communication (URLLC) application requirement issues, this paper focuses on a UAV-based cloud-edge cooperative scheduling system aimed at minimizing the system processing delay while maintaining offloading fairness. We address the optimization of user scheduling, UAV flight trajectory, and task offloading ratio under constraints on the energy consumption and discrete variables. Given the complexity of formulated optimization problem, which involves mixed-integer nonlinear programming (MINLP), non-convexity, high-dimensional state space, and continuous action space, we introduce a deep reinforcement learning (DRL)-based computation offloading approach that leverages the deep deterministic policy gradient (DDPG) algorithm. It explores the continuous action space, encompassing variables like UAV flight angle and speed, takes precise actions according to the current state, and ultimately identifies an optimal offloading strategy. Simulation experiments show that the proposed algorithm notably diminishes the processing delay in contrast to other baseline approaches, while they also confirm that the effectiveness of cloud-edge hybrid offloading surpasses that of either single cloud offloading or UAV offloading strategies.**

***Index Terms*—Computation offloading, unmanned aerial vehicles (UAVs), mobile edge computing (MEC), DDPG, delay optimization.**

I. Introduction

With the development and popularization of 5G technology, the terminal user equipments (UEs), such as smartphones and tablets, generate a large amount of data that requires computational processing. Meanwhile, these devices are increasingly running computation-intensive and delay-sensitive applications like maps and navigation, image rendering, cloud gaming, and telemedicine, thereby requiring substantial computing resources and leading to high energy consumption [1]. However, owing to the size and power consumption constraints of UEs, alongside considerations of production costs, mobility, and portability, computing resources are frequently constrained, thereby impacting the user experience. To enhance the processing efficiency of computing tasks, especially for the 5G ultra-reliable and low-latency communication (URLLC) applications, mobile cloud computing (MCC) has been proposed. It enables the UEs to transfer the computing tasks to the cloud which hosts abundant processing resources, thereby reducing the energy usage. Nevertheless, cloud servers are typically located far from UEs, which can lead to significant transmission delays and negatively impact the user experience [2]. Mobile edge computing (MEC) has emerged as an attractive technology to tackle this problem. It allows smart devices to connect to cloud with lower delay by relocating network control, storage, and computation closer to network edge [3].

The application of MEC has shortened the average distance of wireless transmissions, significantly improving the energy efficiency, transmission delay, and reliability [4]-[7]. The work in [4] focused on jointly optimizing the MEC access network selection and the service layout to enhance service quality cost-effectively by intelligently balancing the access delay, communication delay, and service switching costs. Alfakih *et al*. [5] utilized a SARSA algorithm to address resource allocation problems in edge computing, aiming to make optimal offload decisions that minimize system costs. Xiong *et al*. [6] introduced a hybrid integer programming model that leverages a deep reinforcement learning (DRL) framework alongside an attention technique. This approach aims to address the challenge of task assignment in MEC, thereby reducing the processing delay. Dai *et al*. [7] investigated a motion-aware cloud–edge–end collaborative scheduling strategy based on DRL. By defining the optimization objective, they aimed at reducing the maximum delay among various tasks and subsequently reformulated it into a Markov Decision Process (MDP). Following that, they utilized a DRL with a customized reward system, and incorporated NoisyNet to achieve an almost optimal solution. All the aforementioned studies provide edge computing services through setups equipped with computing capabilities, such as base stations and micro

Manuscript received xx, 2024; revised xx, 2024, accepted 20 Feb. 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFE0107900, in part by the the GuangDong Basic and Applied Basic Research Foundation under grant 2024A1515012052, in part by the National Natural Science Foundation of China under grants 61871433, 61828103, 61201255, and 62071306, and in part by the Research Platform of South China Normal University.

H. Chen, H. Cui, J. Wang, and P. Cao are with the School of Electronic Science and Engineering (School of Microelectronics), South China Normal University, Foshan 528225, China (e-mail: 2023025078@m.scnu.edu.cn, cuihaixia@m.scnu.edu.cn, jiahuanwang@scnu.edu.cn, caop@scnu.edu.cn).

Y. He is with the school of Electronic and Information Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: heyejun@126.com).

M. Guizani is with the Machine Learning Department, Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE (email: mguizani@ieee.org).

data centers. However, deploying edge devices in areas with relatively low UE density is generally not commercially viable. Such coverage vulnerabilities can present obstacles in processing tasks produced by UEs in distant regions [8].

Considering the maneuverability and adaptability of unmanned aerial vehicles (UAVs), there has been extensive research on UAV-assisted MEC. Integrating MEC servers with UAVs in UAV-assisted MEC can significantly decrease system cost, including computation delay and energy consumption [9]. Research in this area has mainly focused on task offloading, computing resource allocation, and path planning [10]-[12]. Kota *et al*. [10] examined a UAV-assisted MEC system using hybrid non-orthogonal multiple access (H-NOMA) with the aim of reducing energy consumption for UEs characterized by high computational demands and sensitivity to delays. They achieved this by proposing a delay-based clustering method along with a low-complexity resource distribution algorithm. Zhang *et al*. [11] studied a system wherein the UAV delivers data processing services to multiple smart devices. They proposed a soft actor-critic (SAC) based approach for computation offloading in UAV-assisted MEC to optimize the system's computation offloading service, with the goal of minimizing terminal processing delay. Lin *et al*. [12] introduced a PDDQNLP algorithm that merges DRL with linear programming (LP), in order to maximize the energy efficiency of the UAV and maintaining offloading fairness. Due to the limited computational resources provided by a single UAV and its small communication coverage, it cannot meet the computational needs of multiple UEs. When the number of UEs increases significantly, the user experience will be severely impacted. Therefore, considering the use of multiple UAVs to cooperatively assist, MEC can fully utilize UAV resources and further improve system performance [13]-[14]. In addition, digital twin (DT)-based aerial computing network is also a very promising research direction. With help of DT, it is easy to obtain comprehensive and high-fidelity real-world state information for model training, enabling intelligent and efficient UAV deployment [15]. However, relying solely on either single UAV offloading or cloud offloading cannot fully optimize the system's performance. By considering hybrid offloading, which leverages the strengths of both UAV and cloud server to achieve complementary advantages, the system performance can be further enhanced.

Recently, more and more researchers have turned their attention to cloud-edge hybrid offloading methods [16]-[22]. In [16], a framework was presented for estimating resources and scheduling tasks to execute hybrid workflows on both cloud computing and edge computing systems. This framework utilizes scheduling technology to carry out workflow tasks on a cooperative cloud-edge system, addressing delay-sensitive application issues and improving resource utilization at the layer of edge. Wang *et al*. [17] introduced a collaborative dynamic task scheduling strategy for cloud-edge system, utilizing a hierarchical deep neural networks (DNNs) approach. This approach realizes the collaborative calculation between the cloud and the edge of DNN models, effectively reducing the execution time for services, thus improving overall system application service quality. Sun *et al*. [18] introduced a cloud-edge cooperative scheduling policy that employs a UAV. This strategy employs a lightweight and flexible genetic algorithm (FGA) to optimize task assignment and UAV positioning, with the objective of reducing both energy usage and processing delay through weighted summation. All studies mentioned in [19]-[22] reduced system delay and energy consumption through employing cloud-edge hybrid offloading schemes. However, many challenges remain in current cloud-edge hybrid offloading research and application, such as the complexity of making offloading decisions and task offloading fairness. In a cloud-UAV hybrid offloading system, determining which tasks to offload to UAV or cloud server, and which tasks to process locally, involves complex decision-making problems. Additionally, when the UAV provide computing services for UEs, there may be an imbalance in user scheduling, resulting in unfair task offloading.

To address these challenges, we suggest a UAV-based cloud-edge collaborative task scheduling strategy, wherein the UAV serves as a special edge node, acting as both a computing facility and a wireless access point. As previously mentioned, deploying edge devices in areas with relatively low UE density is generally not cost-effective, so we use the UAV to act as an edge device in these areas. The cloud and UAV collaborate to handle computing tasks generated by UEs, delivering efficient and reliable MEC services to the regions. In summary, our key contributions are outlined below:

1) **Novel hybrid offloading strategy**: Our work proposes a novel hybrid computation offloading strategy based on the deep deterministic policy gradient (DDPG) algorithm. By leveraging division operations, this paper transforms the multi-objective optimization challenge, involving delay and fairness index, into a single-objective optimization task. This simplification not only reduces the complexity of reward function design, but also avoids the conflicts between competing objectives, which is often a challenging issue in multi-objective optimization. Then, we utilize the proposed DDPG algorithm to derive the optimal solutions for user scheduling, UAV flight trajectory, and task offloading ratio between the UAV and the cloud. Consequently, a good balance between delay and fairness is achieved in the offloading strategy.

2) **Model framework and flexibility**: This study considers the time-varying channel status in the time-slotted cloud-edge cooperative scheduling system. Unlike the existing DDPG designs that may be constrained to the specific network structures, our approach can be applied to both two-layer and three-layer architectures, allowing for a versatile comparison of single offloading strategies and hybrid offloading strategies. This adaptability is particularly important when addressing different network environments and user demands.

3) **Fairness consideration:** Most existing research in UAV-assisted MEC primarily focuses on minimizing delay, which often results in imbalanced task distribution among UEs. This can lead to scenarios where some UEs experience significantly lower delays while others suffer from substantial performance degradation. In contrast, our approach systematically integrates fairness into the optimization process, ensuring balanced task offloading across all UEs. By doing so, we prevent the situations where optimizing solely for delay benefits a subset

of UEs at the expense of others. This fairness is achieved by incorporating a fairness index into the reward function of our DDPG-based method, which ensures the equitable resource distribution without sacrificing the system efficiency. Furthermore, our approach not only improves the overall user experience but also enhances the system robustness in scenarios with diverse user demands.

4) **Validating performance superiority**: Through simulation experiments, the effectiveness of the proposed algorithm over baseline approaches is showcased. Furthermore, it is shown that the hybrid offloading outperforms the single cloud offloading or UAV offloading. The optimal task offloading ratio to minimize the system delay under a three-layer architecture is also identified. The simulation results ultimately highlight the impact of considering the offloading fairness on user scheduling, UAV flight trajectory, and fairness index.

The remainder of this paper is organized as follows. Section II describes the system model, then outlines the optimization problem. Section III gives a brief overview of the fundamental principles of DRL and puts forward a DDPG-based computation offloading approach. Section IV provides the simulation experiments. Finally, Section V summarizes the findings and illustrates the conclusion.

II. SYSTEM MODEL

We focus on a cloud-edge cooperative scheduling system that employs a UAV within a three-layer architecture. The system model comprises K UEs, a single UAV outfitted with a nano MEC server, and a cloud server, as depicted in Fig.1. The UAV's flight duration is split into equal time slots. All UEs within the UAV's coverage area have the opportunity to receive computing and communication services from the UAV. In order to avoid interference and data crosstalk among UEs, a periodic time division multiple access (TDMA) protocol is applied [23]. The UEs have the option to perform computing tasks locally, delegate them to the UAV, or transmit them to the cloud. Table I provides an overview of the primary notations and definitions.

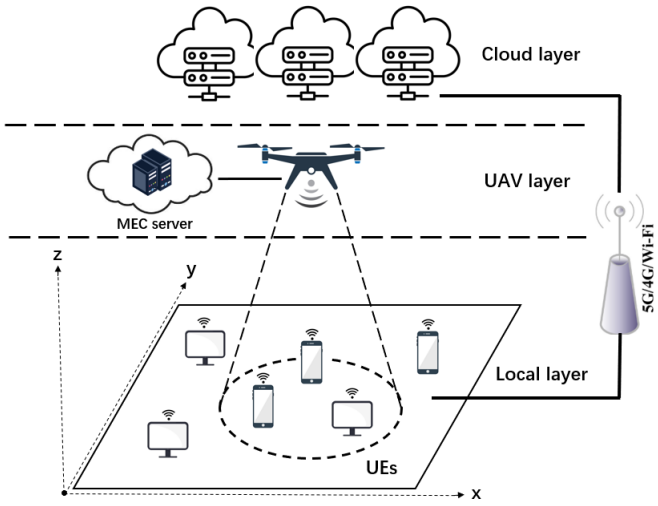


Fig. 1: Cloud-edge cooperative scheduling system.

TABLE I: NOTATIONS AND DEFINITIONS

Notations	Definition
K	The overall count of UEs
I	The count of time slots
T	The UAV's flight cycle
H, L, W	The height, length, and width of the 3D square area
$s_k(i), u(i)$	The spatial vector coordinates of UE k and the UAV
$d_k^{uav}(i)$	The distance between the UE k and the UAV
$d_k^{cloud}(i)$	The distance between the UE k and the cloud
$r_{k,uav}(i)$	The data transmission rate between the UE k and the UAV
$r_{k,cloud}(i)$	The data transmission rate between the UE k and the cloud
$R_k^{uav}(i)$	The task offloading ratio from UE k to the UAV
$R_k^{cloud}(i)$	The task offloading ratio from UE k to the cloud
s	The CPU cycle count necessary for processing each individual byte of data
$\alpha_k(i)$	The binary decision variables about user scheduling
$D_k(i)$	The data volume of UE k
f_{UE}	The processing rate of UE k
f_{UAV}, f_{cloud}	The processing rate of the UAV and cloud
P_{NLoS}	The transmission loss
v_{max}, v_{min}	The UAV's maximum and minimum speeds
C_{max}	The UAV's maximum coverage radius
E_v	The maximum capacity of the battery
net	The potential networks encompass 5G, 4G, and Wi-Fi

A. Network Architecture and Communication Model

We adopt $k \in \{1, 2, ..., K\}$ to represent a collection of UEs. Denote the location of UE k as $s_k = (x_k, y_k)$. A UAV flies at a constant altitude and delivers services to UEs on the ground. We use T to represent a complete flight cycle of the UAV, which is a variable that can be split into I time slots. Utilizing the three-dimensional Cartesian framework for coordinate representation [24], the UAV's coordinate is represented as $u(i) = (x(i), y(i), z)$ at time slot $i \in \{1, 2, ..., I\}$, where z is a fixed value that does not change over time. Similarly, the coordinate of UE k is further denoted as $s_k(i) = (x_k(i), y_k(i))$. To facilitate computation, we mark the UAV's horizontal coordinate as $h(i) = (x(i), y(i))$. Then, the distance between the UE k and the UAV can be expressed as $d_k^{uav}(i) = \sqrt{||h(i) - s_k(i)||^2 + z^2}$. The channel gain for the line-of-sight (LoS) link between the UE k and the UAV is given as [23]

$$g_k^{uav}(i) = \alpha_0 [d_k^{uav}(i)]^{-\delta} = \frac{\alpha_0}{[d_k^{uav}(i)]^{\delta}}, \tag{1}$$

where α_0 indicates the channel gain at a standard distance of $d = 1m$, and δ is the path-loss factor. Since we adopt a LoS channel model in our framework, the path-loss factor δ is set to 2, which is a common value under free-space conditions [18]. This assumption is generally reasonable for communication between the UAV and UEs, as UAVs typically operate at altitudes where LoS transmission is prevalent. Based on Shannon's theorem, the data transmission rate between the UE k and UAV is denoted as

$$r_{k,uav}(i) = B_u \log_2(1 + \frac{P_{up} g_k^{uav}(i)}{\sigma^2 + f_k(i) P_{NLoS}}), \tag{2}$$

where P_{up} represents the transmission power for UE k's uplink. σ^2 indicates the noise power. B_u denotes the transmission bandwidth between the UE k and the UAV. P_{NLoS} denotes

the transmission loss, which occurs when the wireless signal is blocked by obstacles during transmission, resulting in the signal cannot be transmitted to the receiving end in a straight line [25]. $f_k(i)$ is the indicator for whether any blockage exists between the UAV and the UE k (i.e., 1 signifies the presence of blockage, and 0 denotes no blockage) [26].

Moreover, we assume that the cloud server has a fixed position $q_c = (x_c, y_c, z_c)$ that does not change over time slot i. Thus, the distance between the cloud server and the UE k can be expressed as $d_k^{cloud}(i) = \sqrt{[x_c - x_k(i)]^2 + [y_c - y_k(i)]^2 + z_c^2}$. Then, we can further obtain the channel gain between the cloud server and the UE k as

$$g_k^{cloud}(i) = \alpha_0 [d_k^{cloud}(i)]^{-\xi} = \frac{\alpha_0}{[d_k^{cloud}(i)]^{\xi}}, \tag{3}$$

where ξ represents the path-loss factor. Accordingly, the data transmission rate between the cloud server and the UE k can be expressed as

$$r_{k,cloud}(i) = B_c \log_2(1 + \frac{P_{up} g_k^{cloud}(i)}{\omega^2 + f_k(i) P_{NLoS}}), \tag{4}$$

where B_c represents the transmission bandwidth between the UE k and the cloud server and ω^2 denotes the noise power.

B. Computation Model

We presume all UEs move randomly within a square area at a low speed, and a partial offloading policy is applied to a computational task of size $D_k(i)$ produced by UE k at time slot i. The UE k needs to develop an optimal offloading strategy based on the system state (e.g., the distance between the user, the UAV, and the cloud server) and complete the task within the duration of the time slot. We use $R_k^{uav}(i) \in [0, 1]$ to represent the task offloading ratio from UE k to the UAV, and $R_k^{cloud}(i) \in [0, 1]$ indicates the task offloading ratio from UE k to the cloud. Thus, $(1 - R_k^{uav}(i) - R_k^{cloud}(i))$ indicates the percentage of tasks that remain locally processed.

1) *Local Processing* : After the partial offloading policy is executed on the UE k, the remaining tasks are processed locally. Thus, we solely focus on the local processing time of the remaining tasks, which is represented as

$$T_k^{local}(i) = \frac{(1 - R_k^{uav}(i) - R_k^{cloud}(i)) D_k(i) s}{f_{UE}}, \tag{5}$$

where f_{UE} denotes the processing rate of UE k. $D_k(i)$ denotes the computing tasks' size of UE k, and s denotes the CPU cycle count necessary for processing each individual byte of data. Correspondingly, the local energy consumption is listed as

$$E_k^{local}(i) = (1 - R_k^{uav}(i) - R_k^{cloud}(i)) D_k(i) s \cdot e_k^{local}, \tag{6}$$

where e_k^{local} denotes the energy utilized for each computing cycle.

2) *Edge Computing* : In the i-th time slot, the coordinates of the UAV flying from its current position $u(i) = (x(i), y(i), z)$ to the next position can be expressed as

$$u(i+1) = (x(i) + v(i) cos\theta(i) t_{fly}, y(i) + v(i) sin\theta(i) t_{fly}, z), \tag{7}$$

at a speed $v(i) \in [0, v_{max}]$ and an angle $\theta(i) \in [0, 2\pi]$. t_{fly} indicates the flight time. The energy consumed by this flight can be expressed as

$$E_k^{fly}(i) = \beta ||v(i)||^2, \tag{8}$$

where $\beta = 0.5 M_{UAV} t_{fly}$ [27] and M_{UAV} represents the UAV's payload.

The computing tasks generated by the UE k are offloaded to UAV through Wi-Fi. The delay of this layer comprises the computation delay and transmission delay of tasks on the UAV. The output size of the UAV's computation is generally tiny enough to be ignored [27]. Therefore, we disregard the downlink's transmission delay. The execution time can be obtained as

$$T_k^{uav}(i) = \frac{R_k^{uav}(i) D_k(i)}{r_{k,uav}(i)} + \frac{R_k^{uav}(i) D_k(i) s}{f_{UAV}}, \tag{9}$$

In this layer, the communication and computation energy consumption can be represented as

$$E_{k,\text{uav}}^{\text{trans}}(i) = R_k^{\text{uav}}(i) D_k(i) \cdot e_k^{w}, \tag{10}$$

$$E_{k,\text{uav}}^{\text{comp}}(i) = R_k^{\text{uav}}(i) D_k(i) s \cdot e_k^{\text{uav}}, \tag{11}$$

where f_{UAV} represents the processing rate of the UAV. e_k^{uav} indicates the energy consumed for each computing unit of a task on the UAV from the UE k. e_k^{w} represents energy consumption for each unit of data in the Wi-Fi network [18].

3) *Cloud Computing* : Through 5G, 4G, and Wi-Fi networks, UEs can offload their tasks to the cloud. The total delay in this layer encompasses the time for data uploading, calculation, and downloading the calculation results. The download time for the calculation results can be ignored compared to the data upload time, since the data size for calculation results is very small [6]. In this layer, we can describe the execution time as

$$T_k^{cloud}(i) = \frac{R_k^{cloud}(i) D_k(i)}{r_{k,cloud}(i)} + \frac{R_k^{cloud}(i) D_k(i) s}{f_{cloud}}, \tag{12}$$

Accordingly, the energy used for transmission and data processing in this layer can be described as

$$E_{k,cloud}^{trans}(i) = R_k^{cloud}(i) D_k(i) \cdot e_k^{net}, \tag{13}$$

$$E_{k,cloud}^{comp}(i) = R_k^{cloud}(i) D_k(i) s \cdot e_k^{cloud}, \tag{14}$$

where f_{cloud} denotes the processing rate of the cloud. e_k^{cloud} is the energy used each unit of computing in the cloud. Because there is enough power on the cloud, this parameter is set to zero [18]. e_k^{net} is the the energy usage each unit data for Wi-Fi, 5G, and 4G networks.

C. Problem Description and Formulation

Our objective is to minimize the maximum processing delay across all UEs through optimizing the task offloading strategy under the three-layer architecture (local, UAV, and cloud). However, minimizing processing delay may lead to issue with unbalanced user scheduling. The UAV tends to stay close to certain UEs during one flight cycle, resulting in other UEs consistently out of the UAV's coverage range. Consequently, these UEs are unable to access the computational services

provided by the UAV and must either offload tasks to the cloud or handle them locally. In order to address the issue of user scheduling unbalance, we consider introducing Jain's fairness index to ensure offloading fairness between the UAV and UEs [28].

Jain's fairness index is commonly used to measure fairness of resource allocation, with values ranging from 0 to 1, where the value closer to 1 indicates a more fair distribution. In the context of UAV-assisted MEC, the fairness of user scheduling is crucial. Since a UAV can only establish communication with a limited number of UEs within one flight cycle, if the UAV favors certain UEs too much, other UEs will be left outside the UAV's coverage area and unable to receive fair computing resources. Additionally, the fairness of offloading directly affects the balanced processing of all UEs' computing tasks, thereby improving the overall system performance and user experience.

Moreover, Jain's fairness index is a widely used fairness metric, suitable for various resource allocation scenarios [28]. It simply and effectively reflects the balance of resource distribution within a system. In our study, we place Jain's fairness index in the denominator of objective function because we aim to minimize the data processing delay while keeping the fairness index as close to 1 as possible. By minimizing the ratio of delay to the fairness index, we achieve the goal of reducing delay while ensuring offloading fairness. We give the expression for the fairness index as follows

$$f(i) = \frac{\left(\sum_{k=1}^{K}\sum_{i'=1}^{i}\alpha_k(i)\right)^2}{K\sum_{k=1}^{K}\left(\sum_{i'=1}^{i}\alpha_k(i)\right)^2}, \tag{15}$$

where $\alpha_k(i) = \{0, 1\}$ is used to signify the UE's offloading indicator. If $\alpha_k(i) = 1$, tasks at UE k can be offloaded to UAV for processing. Otherwise, tasks have to be handled locally or offloaded to the cloud.

From the analysis above, we aim at minimizing the maximum processing delay across all UEs, while also ensuring offloading fairness between the UAV and UEs. Then, the binary decision variables about user scheduling $\alpha = \{\alpha_k(i)\}$, UAV 3D trajectory $U = \{u(i)\}$, task offloading ratio between the UAV and the cloud $R = \{R_k^{uav}(i), R_k^{cloud}(i)\}$ are jointly optimized. Hence, we can describe the optimization problem in this paper as follows

$$\min_{\{\alpha,U,R\}} \sum_{i=1}^{I}\sum_{k=1}^{K}\frac{\alpha_k(i)}{f(i)} max\{T_k^{local}(i), T_k^{uav}(i), T_k^{cloud}(i)\}, \tag{16}$$

$$s.t. \quad \alpha_k(i) \in \{0,1\}, \forall i,k, \tag{16a}$$

$$\sum_{k=1}^{K}\alpha_k(i) = 1, \forall i, \tag{16b}$$

$$0 \le R_k^{cloud}(i), R_k^{uav}(i) \le 1, \forall i,k, \tag{16c}$$

$$0 \le R_k^{cloud}(i) + R_k^{uav}(i) \le 1, \forall i,k, \tag{16d}$$

$$\alpha_k(i)\sqrt{||h(i) - s_k(i)||^2} \le C_{max}, \forall i,k, \tag{16e}$$

$$s_k(i) \in \{(x_k(i), y_k(i)) | x_k(i) \in [0, L], y_k(i) \in [0, W]\} \quad \forall i,k, \tag{16f}$$

$$h(i) \in \{(x(i), y(i)) | x(i) \in [0, L], y(i) \in [0, W]\}, \forall i, \tag{16g}$$

$$\sum_{i=1}^{I}(E_{k,uav}^{comp}(i) + E_k^{fly}(i)) \le E_v, \forall k, \tag{16h}$$

$$\sum_{i=1}^{I}(E_k^{local}(i) + E_{k,uav}^{trans}(i) + E_{k,cloud}^{trans}(i)) \le E_k, \forall k, \tag{16i}$$

$$\sum_{i=1}^{I}\sum_{k=1}^{K}\alpha_k(i)D_k(i) = D, \tag{16j}$$

where the binary constraints $(16a)$ and $(16b)$ ensure that the UAV serves one user during a time slot. Constraint $(16c)$ and constraint $(16d)$ specifie the range of task offloading ratio for the cloud and UAV while ensuring that the sum of the two does not exceed 1. Constraint $(16e)$ guarantees that the UAV can only dispatch UEs within its coverage area. Constraint $(16f)$ and constraint $(16g)$ guarantee that the UAV and UEs remain within the designated area. Constraint $(16h)$ states that the UAV's energy usage cannot surpass the battery's maximum capacity during a flight cycle. Constraint $(16i)$ indicates that the total energy consumption of each UE must not exceed its available energy. Constraint $(16j)$ specifies that all computing tasks generated by UEs must be processed within a period.

To ensure the compliance with both short-term and long-term constraints during the optimization process, we will implement several mechanisms in the following. First, we use a tanh activation function to constrain the offloading ratios within a reasonable range [29], and ensure the sum of the UAV and cloud server offloading ratios does not exceed 1. If exceeded, the UE k will remain in local computation, driving the optimization to meet the constraint $(16d)$. Second, the horizontal distance between the UAV and UE k is considered, and if it exceeds the UAV's maximum coverage C_{max}, the user cannot be scheduled. We also check the movement range of the UAV and UE k during state updates to ensure they do not exceed the permitted area. Additionally, if the UAV's remaining energy is insufficient for computation, the offloading ratio of UAV is set to zero, shifting to a cloud-based offloading scheme, so that it can encourage the optimization process and better adhere to the UAV's energy constraint $(16h)$.

III. ALGORITHM DESIGN

We utilize DRL to address the optimization problem (16). Compared to traditional algorithms, DRL requires fewer computational resources and can circumvent combinatorial optimization problems. This can significantly enhance the efficiency of our model training process. First, we approach the optimization problem from an MDP perspective, defining the state, action, and reward function. Second, we use the DQN to tackle the high-dimensional state space challenges, then propose the DDPG for managing continuous action spaces. Finally, we provide the implementation process for training the algorithms.

A. State, Action, and Reward Definition

1) *State Space* : In our proposed cloud-edge cooperative scheduling system, the state space is determined collectively through K UEs, a single UAV, a cloud server as well as the surrounding environment. The MDP state is described by a set of states s_i that illustrate the feasible setups of the agent. Accordingly, the system's state vector is listed as

$$s_i = (s_k(i), u(i), d_k^{cloud}(i), D_k(i), D_{remain}(i), E_{battery}(i)), \tag{17}$$

where $s_k(i)$ and $u(i)$ represent the location information of the UE k and the UAV, respectively. $d_k^{cloud}(i)$ indicates the distance between the cloud and UE k. The size of the tasks generated randomly by UE k is represented by $D_k(i)$. Moreover, $D_{remain}(i)$ represents the tasks' size remaining for the entire system to process. $E_{battery}(i)$ indicates the UAV's remaining battery power during time slot i. It's worth mentioning when $i = 1$, $E_{battery}(i) = E_v$ and $D_{remain}(i) = D$.

2) *Action Space* : The agent aims at mapping the observed state space to the action space. In simpler terms, the agent takes decisions according to its present state and the surrounding environment. In our system design, the agent handles the offloading decisions for the UE k to be served in each time slot i. Additionally, the agent optimizes the UAV's flight trajectory by adjusting the flight speed and angle to achieve offloading fairness. Thus, the action space consists of selecting which UE k to serve, the UAV's speed and angle, the offloading ratio between the UAV and cloud. Then, the action vector is represented as

$$a_i = (k(i), v(i), \theta(i), R_k^{uav}(i), R_k^{cloud}(i)), \tag{18}$$

Note that the output of the DDPG algorithm is continuous action, so we need to discretize the parameter $k(i) \in [0, K]$, which has an integer data type. Specifically, if $k(i) \neq 0$, then $k'(i) = \lceil k(i) \rceil$, and if $k(i) = 0$, then $k'(i) = 1$. Here, $\lceil \cdot \rceil$denotes the ceiling operation for integers [1]. We scale the output speed, angle, and task offloading ratio according to the actual action range, then output the scaled action to the environment, where $v(i) \in [0, v_{max}]$, $\theta(i) \in [0, 2\pi]$, $R_k^{uav}(i)$ and $R_k^{cloud}(i) \in [0, 1]$. The five parameters mentioned above will be jointly optimized to minimize the maximum processing delay among UEs while ensuring offloading fairness between the UAV and UEs.

3) *Reward Function* : Generally speaking, the objective function has a connection to the critic network's reward function. We aim at maximizing the cumulative discounted reward in reinforcement learning (RL). But the proposed cloud-edge cooperative scheduling system's goal is to minimize system delay while ensuring offloading fairness, which represents an inverse goal compared to RL. Hence, the reward function's value must be negatively correlated with the objective function's value. The agent takes action a_i at time slot i to engage with the surrounding environment in a specific state s_i, and as it enters the new state s_{i+1}, it simultaneously obtains a reward $r(s_i, a_i)$. We aim at maximizing the reward by minimizing the ratio of the maximum processing delay to the fairness index in problem (16), as follows

$$r_i = r(s_i, a_i) = -\psi(i), \tag{19}$$

where the ratio of the processing delay to the fairness index can be denoted as

$$\psi(i) = \sum_{k=1}^{K} \frac{\alpha_k(i)}{f(i)} max\{T_k^{local}(i), T_k^{uav}(i), T_k^{cloud}(i)\}, \tag{20}$$

and if $k = k'$, it indicates that the UE k is scheduled by the UAV, then $\alpha_k(i) = 1$. Else, $\alpha_k(i) = 0$.

B. Markov Decision Process

If a random process's next state depends only on the current state but is unaffected by previous states, then it is said to exhibit the Markov property. Any game or control task that possesses Markov property can be termed a Markov Decision Process (MDP) [30]. The MDP model makes RL problems significantly simpler because the agent's actions depend solely on the current state, while the reward and the new state are determined by both the present state and the action undertaken. Thus, modeling RL problems as MDPs provides a more precise description for the process of the agent interacting with its surrounding environment and taking decisions.

We convert the proposed optimization problem to MDP and define MDP as a quadruple $(\mathcal{S}, \mathcal{A}, p(.,.), r)$, where $\mathcal{S}$ and $\mathcal{A}$ indicate the sets of potential states and actions for the agent, respectively. We employ a policy $\pi(s_i, a_i)$ to associate the current state with the appropriate action, where $s_i \in \mathcal{S}$ and $a_i \in \mathcal{A}$. Under the guidance of policy π, the agent in the state s_i chooses the corresponding action a_i, then moves to the next state s_{i+1} with a probability of transition $p(s_{i+1}|s_i, a_i)$, while acquires an immediate reward $r_i = r(s_i, a_i)$. The primary aim of the agent is to find an optimal strategy π^*, thereby maximizing the anticipated cumulative discounted reward, as follows

$$R_i = \sum_{i=0}^{\infty} \gamma^i r_i, \tag{21}$$

where $\gamma \in [0, 1]$ is the discount factor. Under the strategy π, since the state value function $V_\pi(s_i)$ denotes the discounted rewards' anticipated sum, the expression of $V_\pi(s_i)$ of the agent in the state s_i can be concluded as

$$V_\pi(s_i) = \mathbb{E}_\pi[R_i|s_0 = s_i] = \mathbb{E}_\pi[(\sum_{i=0}^{\infty} \gamma^i r_i)|s_0 = s_i], \tag{22}$$

As mentioned earlier, we use $p(s_{i+1}|s_i, a_i)$ to characterize the likelihood of transitioning to the new state s_{i+1} after the current state s_i performs the corresponding action a_i. Then according to the Bellman equation, we can transform the state value function into a time-difference form and expressed as

$$\begin{aligned} V_\pi(s_i) &= \mathbb{E}_\pi[(\sum_{i=0}^{\infty} \gamma^i r_i)|s_0 = s_i] \\ &= \mathbb{E}_\pi[(r(s_i, a_i) + \sum_{i=1}^{\infty} \gamma^i r_i)|s_0 = s_{i+1}] \\ &= r(s_i, a_i) + \gamma \sum_{s_{i+1}} p(s_{i+1}|s_i, a_i) V_\pi(s_{i+1}), \end{aligned} \tag{23}$$

To achieve our objective of determining the optimal strategy π^*, the optimal state value function $V_{\pi^*}(s_i)$ guides the

identification of the best action a_i^* in each state. We describe $V_{\pi^*}(s_i)$ as follows.

$$V_{\pi^*}(s_i) = \max_{\pi}[r(s_i, a_i) + \gamma \sum_{s_{i+1}} p(s_{i+1}|s_i, a_i)V_{\pi^*}(s_{i+1})], \tag{24}$$

The solution $\pi^*(s_i)$ satisfying equation (18) is called the optimal policy, and the optimal action a_i^* corresponding to the state s_i can be expressed as

$$a_i^* = \underset{a_i}{argmax}\ V_\pi(s_i, a_i), \tag{25}$$

C. Improved DQN Algorithm

The Q-learning algorithm employs a Q-table to represent state-action values, sometimes called Q-values. Each cell of this Q-table holds an approximated Q-value corresponding to a particular state-action pair. When the agent engages with the environment and gains rewards, the algorithm repeatedly updates these Q-values until they reach their optimal values [31]. However, Q-table is only effective for managing simple problems with few states. As the system model becomes more complex, employing a memory table becomes impractical.

To address the challenges encountered in Q-learning, we employ a Q-function instead of a Q-table. Since neural networks are very good at modeling complex functions, we can use an improved approach employing deep Q network (DQN), where DNNs are used to build the Q-function [32]. This function maps the state to the Q-values of all possible actions from that state, and the optimal Q-values $Q_*(s, a)$ are approximately derived through utilizing the updated DNN parameter θ, that is, $Q_*(s, a) \approx Q(s, a; \theta)$.

The DQN structure includes two networks: the target network and the Q network, along with a component referred to as experience replay. An agent that can generate optimal state-value pairs after training is termed a Q network. To enhance the robustness of the training model, the target network and experience replay are employed [33]. First, the specific practice of experience replay is that in each time step i, a DQN agent stores its interactive experience tuple (s_i, a_i, r_i, s_{i+1}) into the replay pool. After that, a random mini-batch of M samples (s_n, a_n, r_n, s_{n+1}) is chosen from the replay pool to update the DNN's parameters. The second technique for stabilization involves employing a target network structured identically to the Q network. The specific method entails keeping the target network's parameters constant for a defined duration, then training the Q network using gradient descent during this time frame, and finally the parameters obtained from the Q network are used to regularly update the target network.

In each iteration, DQN selects the optimal action according to the target network's weight parameter θ_n^-, and as mentioned earlier, θ_n^- is periodically adjusted according to the counterpart θ_n learned by the Q network. Thus, $\theta_n^- = \theta_{n-Z}$, which means that the target network updates its weights each Z time intervals. Then, we can obtain the expression of the target Q-value y_n as follows

$$y_n = r_n + \gamma \max_{a_{n+1}} Q(s_{n+1}, a_{n+1}|\theta_n^-), \tag{26}$$

Algorithm 1 A DQN-based computation offloading training algorithm

1: **Initialize:** the Q network weight θ and the target network weight $\theta^- = \theta$;
2: **Initialize:** the experience replay pool to capacity R and mini-batch size M;
3: **for** each episode **do**
4: Reset environment, and obtain the initial state s_0;
5: **for** time slot = [1,...,i,...,I] **do**
6: Select a_i with a random probability ϕ as
7: **if** $\phi \leq \epsilon$ **then**
8: select an action a_i randomly;
9: **else**
10: choose $a_i = \underset{a}{argmax}\, Q(s_i, a_i; \theta)$;
11: **end if**
12: Perform action a_i, then receive the reward r_i and
13: move to the next state s_{i+1};
14: Store the experience tuple (s_i, a_i, r_i, s_{i+1}) into the
15: experience replay pool;
16: A mini-batch of M samples (s_n, a_n, r_n, s_{n+1}) is
17: selected from the experience replay pool;
18: Set $y_n = r_n + \gamma \max_{a_{n+1}} Q(s_{n+1}, a_{n+1}|\theta_n^-)$;
19: Update the Q network weight θ by minimizing
20: the loss function $L(\theta)$ as
21: $L(\theta) = \frac{1}{M}\sum_n (y_n - Q(s_n, a_n|\theta_n))^2$;
22: The loss function $L(\theta)$ is used for gradient descent
23: of the weight θ deflection;
24: The target network weight θ_n^- are updated using
25: the parameter θ_n;
26: **end for**
27: **end for**

Additionally, we minimize the loss function $L(\theta)$ to train the Q-function toward the target value. The expression of $L(\theta)$ can be shown as

$$\begin{aligned} L(\theta) &= \mathbb{E}[(y_n - Q(s_n, a_n|\theta_n))^2] \\ &= \frac{1}{M}\sum_n (y_n - Q(s_n, a_n|\theta_n))^2, \end{aligned} \tag{27}$$

Algorithm 1 describes the DQN-based training process for computation offloading. First, we initialize the weight parameter of the target network as well as the Q network, that is, $\theta^- = \theta$. Determine the experience replay pool's capacity and specify the size of mini-batch samples for training. Second, we use the ϵ-greedy strategy for action selection at each step, the agent has a probability ϵ of selecting an action randomly, and with a probability $(1-\epsilon)$ to opt for the action that exhibits the maximum Q-value across all available options. Then, the agent performs the selected action to engage with the environment, obtaining a reward while proceeding to the next state. And the transition memory tuple (s_i, a_i, r_i, s_{i+1}) is kept into the replay pool. Finally, a mini-batch of M samples (s_n, a_n, r_n, s_{n+1}) is selected from the replay pool and inputs it into Q network. After the whole training process above, we can compute the target Q-value y_n using the formula in line 19. Subsequently,

we update the Q network's parameters θ through minimizing $L(\theta)$. To ensure convergence of the Q-function to its optimal value, we employ gradient descent.

D. Improved DDPG Algorithm

DQN pioneers the fusion of deep learning (DL) and RL, effectively mastering the technique of directly learning control strategies from high-dimensional state spaces. However, DQN is unable to produce the action value function for individual actions in continuous action spaces. A straightforward solution to tackle the challenge of continuous action spaces, as mentioned previously, involves discretizing the action space [32]. However, the exponential growth of the action space with increasing degrees of freedom makes this approach impractical for most control tasks.

To overcome the limitations of DQN, we utilize the DDPG algorithm, which is an offline DRL algorithm specifically designed to tackle continuous control problems [34]. The DDPG algorithm employs a twin neural network for the value function as well as the policy function, enabling a more stable learning process and faster convergence. As illustrated in Fig. 2, the DDPG algorithm comprises four networks: the actor network $\mu(s|\theta^{\mu})$, the critic network $Q(s,a|\theta^{Q})$, and copies of these two networks, namely two independent target networks $\mu'(s|\theta^{\mu'})$ and $Q'(s,a|\theta^{Q'})$. Similar to DQN, the update of θ^{Q} by $Q(s,a|\theta^{Q})$ can be expressed as

$$\begin{aligned} L(\theta^{Q}) &= \mathbb{E}_{\mu'}[(y_n - Q(s_n,a_n|\theta^{Q}))^2] \\ &= \frac{1}{M}\sum_{n}(y_n - Q(s_n,a_n|\theta^{Q}))^2, \end{aligned} \tag{28}$$

where M represents a mini-batch samples' size, and y_n represents the target Q-value. The expression of y_n as follows

$$y_n = r_n + \gamma Q'(s_{n+1}, \mu'(s_{n+1}|\theta^{\mu'})|\theta^{Q'}), \tag{29}$$

With the loss function $L(\theta^{Q})$, the parameter θ^{μ} of the online actor network can be adjusted by calculating the sample strategy gradient $\bigtriangledown_{\theta^{\mu}}\mu|s_n$ based on the standard backward propagation method [35],

$$\begin{aligned} \bigtriangledown_{\theta^{\mu}}\mu|s_n &= \mathbb{E}_{\mu'}[\bigtriangledown_a Q(s_n,\mu(s_n|\theta^{\mu})|\theta^{Q})\bigtriangledown_{\theta^{\mu}}\mu(s_n|\theta^{\mu})] \\ &= \frac{1}{M}\sum_{n}(\bigtriangledown_a Q(s_n,\mu(s_n|\theta^{\mu})|\theta^{Q})\bigtriangledown_{\theta^{\mu}}\mu(s_n|\theta^{\mu})), \end{aligned} \tag{30}$$

Algorithm 2 describes the DDPG-based training process for computation offloading. First, we initialize the parameters θ^{μ} and θ^{Q} of the actor network $\mu(s|\theta^{\mu})$ and the critic network $Q(s,a|\theta^{Q})$, respectively. The parameters θ^{μ} and θ^{Q} are copied to the parameters $\theta^{\mu'}$ and $\theta^{Q'}$ of the corresponding two target networks. Next, initialize the experience replay pool's capacity, while specify the size of mini-batch samples for training. Second, the agent performs the action a_i to engage with the surrounding environment, receiving a reward r_i, then moving to the new state s_{i+1}. It's important to mention that to ensure the agent fully explores the state space, we need to add behavior noise to the action space, i.e., $a_i = \mu(s_i|\theta^{\mu}) + n_i$, where n_i follows a Gaussian distribution $n_i \sim \mathbb{N}(\mu_e, \sigma_{e,i}^2)$, with μ_e representing the mean, $\sigma_{e,i}^2$ as the standard deviation. The next step is to store the state transition sequence (s_i, a_i, r_i, s_{i+1}) in the replay pool R, and randomly sample M transition sequences (s_n, a_n, r_n, s_{n+1}) from it as a small batch of training data for the actor network as well as the critic network. Finally, the agent updates θ^{μ} and θ^{Q} by minimizing the loss function $L(\theta^{Q})$ while calculating the sample strategy gradient $\bigtriangledown_{\theta^{\mu}}\mu|s_n$. Additionally, $\theta^{\mu'}$ and $\theta^{Q'}$ are updated via soft updates [36].

Algorithm 2 A DDPG-based computation offloading training algorithm

Initialize: the network weights θ^{μ} and θ^{Q} for the actor network $\mu(s|\theta^{\mu})$ and critic network $Q(s,a|\theta^{Q})$;
Initialize: the weights of the target network $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$, respectively;
Initialize: the experience replay pool to capacity R and mini-batch size M;
for each episode **do**
  Reset environment, and obtain the initial state s_0;
  for time slot = [1,...,i,...,I] **do**
    Get an action with the current strategy and the behavior noise, $a_i = \mu(s_i|\theta^{\mu}) + n_i$;
    Perform action a_i, then receive the reward r_i and move to the next state s_{i+1};
    Store the experience tuple (s_i, a_i, r_i, s_{i+1}) into the experience replay pool;
    A mini-batch of M samples (s_n, a_n, r_n, s_{n+1}) is selected from the experience replay pool;
    Set $y_n = r_n + \gamma Q'(s_{n+1}, \mu'(s_{n+1}|\theta^{\mu'})|\theta^{Q'})$;
    Update the critic network weight θ^{Q} by minimizing the loss function $L(\theta^{Q})$ as
    $L(\theta^{Q}) = \frac{1}{M}\sum_{n}(y_n - Q(s_n,a_n|\theta^{Q}))^2$;
    The actor network weight θ^{μ} is updated by calculating the sample strategy gradient $\bigtriangledown_{\theta^{\mu}}\mu|s_n$;
    The target network weights $\theta^{\mu'}$ and $\theta^{Q'}$ are updated via soft updates as
    $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$,
    $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$;
  end for
end for

IV. RESULTS AND ANALYSIS

This section primarily show the efficacy of the proposed DDPG algorithm in cloud-edge cooperative scheduling system through simulation experiments. First, we describe the simulation parameter settings and network structure. Second, we contrast the DDPG algorithm's delay index with those of five other baseline methods. Additionally, we compare the delay of the DDPG in a two-layer architecture versus a three-layer architecture. Third, we determine the approximately optimal task offloading ratio that minimizes the system's latency. Finally, we discussed the offloading fairness between the UAV and UEs.

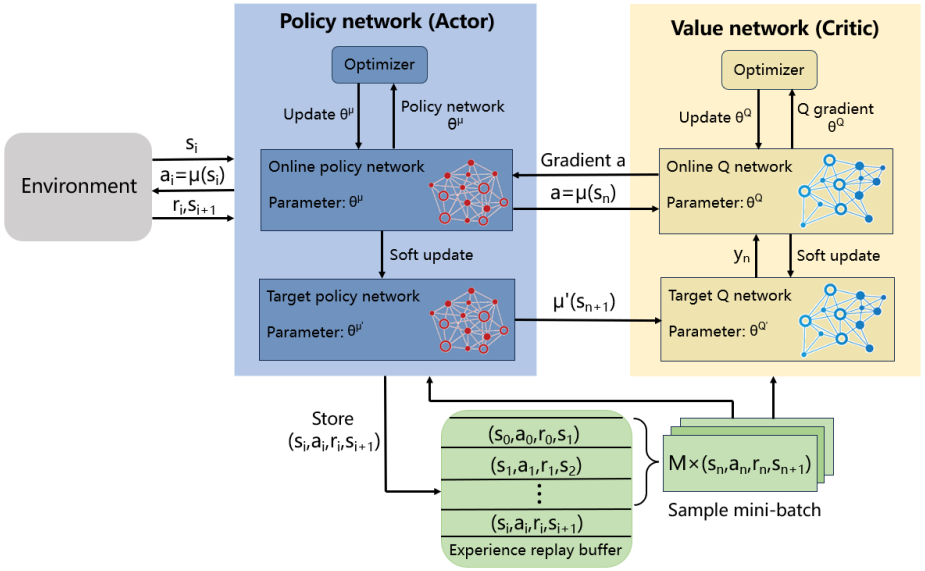


Fig. 2: Our improved DDPG learning process.

A. Experiment Setting

1) *Simulation parameter settings* : In this simulation experiment, there are $K = 6$ UEs distributed randomly within a square region of $L \times W = 100 \times 100$ m^2, while the height of these UEs are set to 0. We assume the UAV maintains a constant altitude of $H = 100$ m over this region. The UAV's weight is $M_{uav} = 9.65$ kg. Its maximum speed is set to $v_{max} = 50$ m/s. The UAV's battery capacity is set to $E_v = 500$ KJ, and it must meet the energy consumption constraint $(16h)$ during a flight cycle. The bandwidth for UE-UAV and UE-Cloud is set to $B_u = 1$ MHz and $B_c = 2$ MHz, respectively. The entire duration, $T = 120$ s, is split into $I = 40$ time slots. For a standard distance of 1 m, the channel gain is $\alpha_0 = -50$ dB. The penetration loss of non-line-of-sight connectivity, $P_{NLoS} \in [20, 30]$ dB. We randomly generate a value for the parameter $f_k(i)$ of each UE in the simulation environment to represent whether the communication between the UE k and the UAV is affected by obstacles. If the signal is blocked during transmission, then $f_k(i) = 1$. Otherwise, $f_k(i) = 0$. The transmission power of the uplink of UEs P_{up} is uniformly distributed in the range of [0.1, 0.3] W [37]. The processing rate of UEs, UAV and cloud server are set to $f_{UE} \in [0.4, 0.8]$ GHz , $f_{UAV} = 1.2$ GHz and $f_{cloud} = 2.4$ GHz [38]. The detailed system parameters are provided in Table II.

TABLE II: SYSTEM PARAMETER SETTINGS

Parameter	Definition	Value
K	The overall count of UEs	6
T	The duration of a cycle	120s
I	The count of time slots	40
L, W	The side length of the square area	100m
H	The UAV's flying altitude	100m
E_v	The UAV's battery capacity	500KJ
v_{max}	The UAV's maximum speed	50m/s
P_{NLoS}	The transmission loss	[20,30]dB
P_{up}	The transimission power	[0.1,0.3]W
D_k	The data size of the UE k	[1.5,4]Mbits
f_{cloud}	The processing rate of the cloud	2.4GHz
f_{UAV}	The processing rate of the UAV	1.2GHz
f_{UE}	The processing rate of the UE k	[0.4,0.8]GHz
α_0	The channel power gain	-50dB
s	The CPU cycles needed for each bit	1000 cycles/bit

2) *Network structure* : Fig. 2 depicts the DDPG algorithm's network structure employed in this paper. The DDPG approximates the action value function using an actor network as well as a critic network. The actor network comprises input, hidden, and output layers, with dimensions of [s.dim], [400, 300, 10] and [a.dim], respectively. Additionally, the critic network has input, hidden, and output layers with dimensions [s.dim + a.dim], [300, 10] and [1], respectively. s.dim represents the dimension of the state, while a.dim represents the action's dimension. According to (17), s.dim = 5 + 4·K, where K represents the overall count of UEs. According to (18), the action space consists of the UE identifier, the UAV's flight speed and angle, as well as the task assignment ratio between the cloud and the UAV. Therefore, a.dim = 5. The ReLU function is employed for the input and hidden layers of the actor network, while the output layers employ the tanh function. The critic network's input layers utilize the matmul function to activate, while the hidden and output layers employ the ReLU function.

The DQN algorithm employs two neural networks (the Q network and the target network) to estimate the Q-values. The Q network is responsible for generating the Q-value estimates for the current state, while the target network is used to compute the target Q-values. The Q network consists of fully connected layers in the input, hidden, and output layers. The input layer has 100 neurons with a ReLU activation function. The hidden layer has 20 neurons with a ReLU activation function as well, and the output layer has a number of neurons equal to the possible combinations of the action space, with a Softmax activation function to output the probability distribution of each action. The structure of the target network

is consistent with the Q network in terms of the number of layers, number of neurons, and activation functions. The random exploration probability in the ϵ-greedy strategy is set to ϵ=0.9. The reward discount factor is set to 0.99.

B. Performance Comparison

1) *Baseline approaches* : To evaluate the effectiveness against the proposed DDPG algorithm, we outline the following four baseline approaches:

— *Local Only* : Without assistance from the UAV and cloud server, all computing tasks generated by UEs are executed locally.

— *Offload UAV Only* : During one flight cycle of the UAV, the computing tasks generated by UEs are entirely transmitted to the UAV for processing.

— *Offload Cloud Only* : The UEs within the square area offload all their generated tasks to the remote cloud for computation.

— *DQN* : The DQN can only handle discrete action spaces. Hence, we need to discretize the UAV's flight speed and angle, as well as the task offloading ratio between the UAV and the cloud. The UAV's flight speed can be expressed as $\mathcal{V} = \{0, 0.1v_{max}, ..., v_{max}\}$, and the angle is defined as $\mathcal{M} = \{0, 0.2\pi, ..., 2\pi\}$. Additionally, the task offloading ratio between the cloud and the UAV can be expressed as $\mathcal{R} = \{0, 0.1, ..., 1.0\}$.

— *DDQN* : The DDQN algorithm improves upon the DQN algorithm by mitigating the overestimation bias of Q-values. Like DQN, DDQN is limited to handling discrete action spaces. Therefore, we use the same method to discretize the UAV's flight speed, flight angle, and task offloading ratio. The key improvement of DDQN over DQN lies in decoupling action selection and action evaluation through the use of two separate Q-networks. This helps to achieve more accurate Q-value updates and better convergence in task offloading decisions.

2) *Evaluating offloading performance* : The contrast in performance between the DDPG algorithm and different baseline approaches is shown in Fig. 3. In this figure, we train three DRL algorithms, DQN, DDQN and DDPG, for 1000 episodes, then observed their convergence with respect to delay. It can be observed that if all the computing tasks are reserved for local processing, it leads to high processing delay. Conversely, offloading all tasks to either UAV or cloud server can significantly lower processing delay. Nevertheless, following the convergence of the DQN, DDQN and DDPG algorithms, the performance of the aforementioned three baseline approaches is notably inferior to that of the DRL algorithms. We find that the DDQN algorithm performs better than the DQN algorithm. This is because DDQN effectively mitigates the issue of Q-value overestimation by using a double Q-network, resulting in better convergence performance in task offloading decisions. Compared to DDQN, our proposed DDPG algorithm demonstrates superior performance in processing delay. Indeed, the limitations of DQN and DDQN in handling continuous action space make it difficult to find the optimal offloading strategy. In contrast, the DDPG effectively searches the continuous action space, enabling it to take precise actions according to the current state, eventually identifying an optimal task offloading strategy.

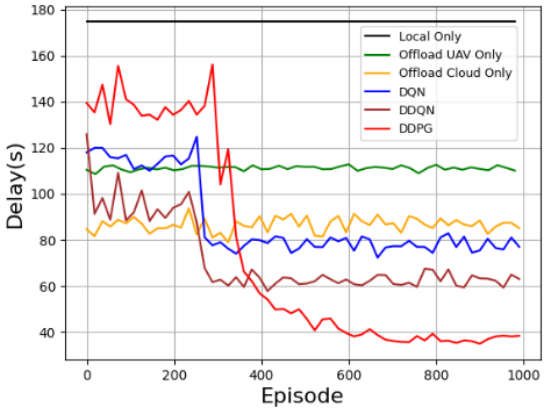


Fig. 3: Performance with task sizes D = 100 Mbits for various algorithms.

Next, we delve into the DDPG's delay performance when combined with the various offloading solutions. These solutions encompass hybrid offloading, cloud offloading, and UAV offloading. As shown in Fig. 4, we can observe that the DDPG with cloud offloading can achieve lower processing delay than the DDPG with UAV offloading. This is because the cloud possesses robust computing power and abundant computing resources, enabling it to swiftly execute computing tasks and provide results. Conversely, despite the relatively close physical proximity between the UAV and UEs, the computing resources and power of the UAV are limited, resulting in comparatively slower processing speeds. Furthermore, the DDPG with hybrid offloading, which integrates both cloud offloading and UAV offloading, can achieve the lowest system delay. This is because hybrid offloading is enabled to fully use the resources from both the cloud and UAV, leveraging their complementary advantages to increase task processing efficiency. Depending on specific scenarios and needs, the distribution of computing resources between the UAV and cloud can be dynamically adjusted to suit different task types and sizes. This flexibility and scalability help reduce delay and enhance system performance.

Leveraging the convergence results of these algorithms, we compare their delays across varying task sizes and UAV flight altitudes, as shown in Fig. 5 and Fig. 6, respectively. From Fig. 5 we can notice that for a given task size, the DDPG algorithm continuously demonstrates the lowest delay compared to the other baseline approaches, with delay increasing as task sizes grow. Since the *Local Only*, *Offload UAV Only*, and *Offload Cloud Only* approaches don't fully use the system's computing resources, while the DQN and DDQN algorithm unable to determine the optimal offloading policy, only the DDPG algorithm can effectively reduce system delay, indicating its superiority. Fig. 6 shows that as the UAV altitude increases, the total system delay rises significantly. This is due to the increased distance between the UAV and UEs, which weakens the channel gain, thereby reducing the data transmission rate and ultimately leading to higher transmission

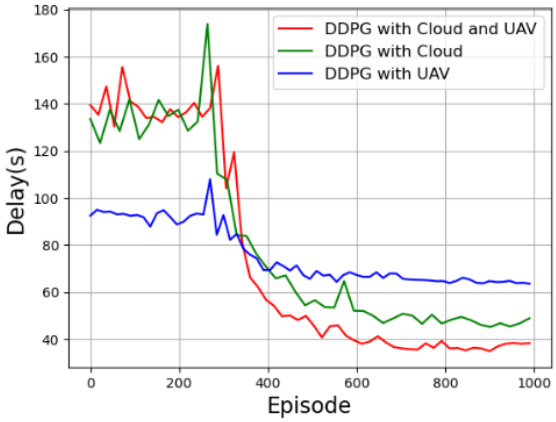


Fig. 4: The delay of the DDPG algorithm when combined with different offloading solutions.

delay. However, our proposed DDPG algorithm still maintains the lowest system delay.

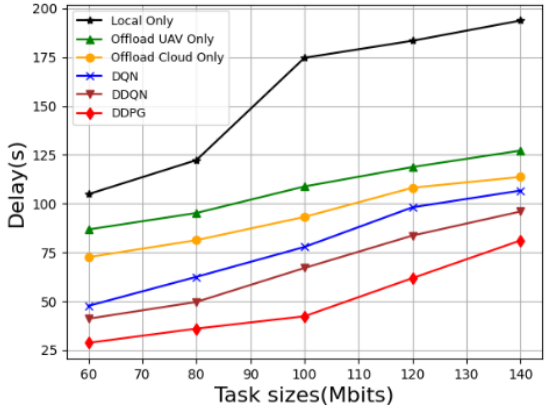


Fig. 5: Delay across different task sizes.

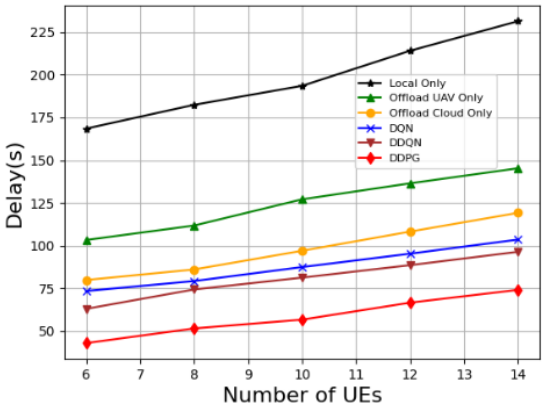


Fig. 7: The impact of the number of UEs on delay.

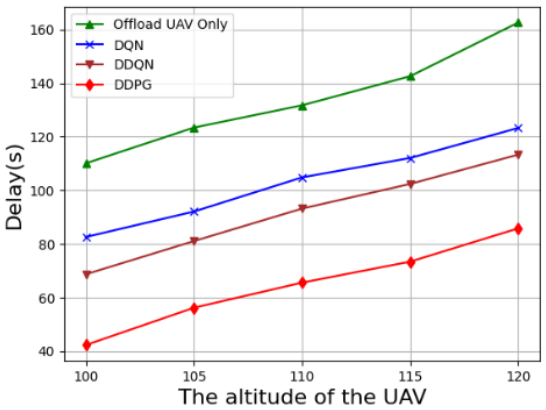


Fig. 6: The impact of the altitude of the UAV on delay.

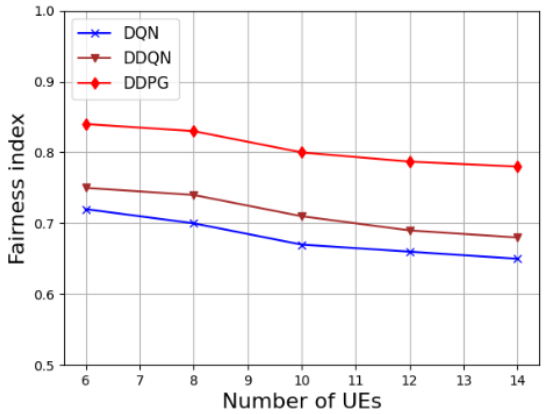


Fig. 8: The impact of the number of UEs on fairness index.

To evaluate the effectiveness of the model in larger network scenarios, we study the impact of different numbers of UEs on system delay and fairness, as shown in Fig. 7 and Fig. 8, respectively. Fig. 7 illustrates a comparison of the total delay for six different methods as the number of UEs increases. The delay increases for all methods as the number of UEs grows. This is due to the fact that the cloud server and UAV need to handle more tasks from additional UEs, which leads to increased computational and communication load. Moreover, as the number of UEs increases, the UAV's throughput during flight approaches the upper limit of its physical and communication resources, ultimately resulting in higher system delay. Fig. 8 presents a comparison of fairness among three DRL algorithms as the number of UEs increases. As more UEs are introduced, maintaining fairness in task offloading becomes increasingly difficult. This is because the increase in the number of UEs results in more UEs being farther away from the UAV, and UEs that are farther away from the UAV usually have to rely on a single cloud offloading strategy. To maximize system rewards, sacrificing some level of fairness in offloading can effectively reduce delay. Based on the above analysis, we observed that while the number of UEs affects both delay and fairness in offloading, our proposed DDPG algorithm still outperforms the baseline methods in terms of handling variations in UE numbers.

Then we discuss the allocation ratio of tasks on the cloud-edge in hybrid offloading. Fig. 9 illustrates the offloading ratio of computing tasks generated by UEs on local, UAV and cloud. According to Fig. 4, the DDPG algorithm starts converging

after 600 training episodes. Thus, the offloading ratio also begins to stabilize after 600 episodes of training as shown in Fig. 9. When over half of the computing tasks generated by UEs are offloaded to the cloud, approximately 30 percent are offloaded to the UAV, while the remaining 10 percent are allocated for local processing, this configuration results in the lowest system delay. We refer to this distribution ratio of tasks as the approximately optimal offloading ratio. Thus, the final offloading strategy is determined by this optimal ratio.

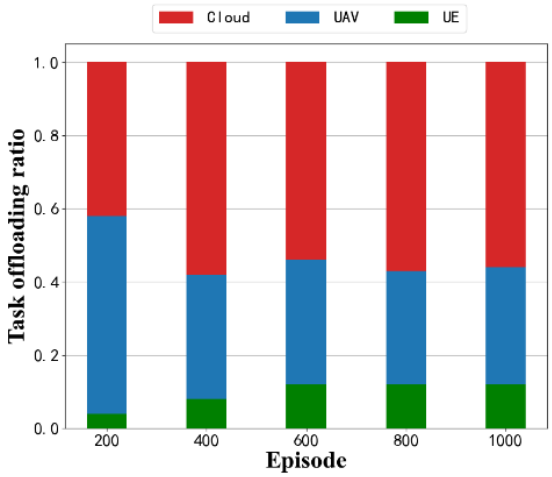


Fig. 9: The task assignment ratio on the cloud-edge in hybrid offloading.

Finally, we delve into the offloading fairness between the UAV and UEs. Fig. 10 - Fig. 12 illustrate the impact of accounting for offloading fairness on user scheduling, fairness index and UAV flight trajectory. Observing Fig. 10, it becomes apparent that without considering offloading fairness, the UAV encounters an unbalanced user scheduling problem. Of the six UEs, UE1 was dispatched up to 17 times during a flight cycle of the UAV, while UE5 and UE6 were never dispatched by UAV. This occurs because, to minimize system delay, the UAV consistently tends to approach certain UEs, consequently leaving other UEs consistently outside the UAV's coverage during a flight cycle, rendering them unable to access the computing services provided by the UAV. On the contrary, the DDPG algorithm that considers offloading fairness tends to balance the number of times the UAV dispatches each UE, thereby enhancing the fairness and efficiency of the UAV computing services.

Accordingly, in Fig. 11, the DDPG algorithm without considering fairness exhibits a poorer fairness index. Throughout a UAV's flight cycle, its fairness index can only reach about 0.4, whereas the DDPG algorithm, when considering fairness, can elevate the fairness index beyond 0.8. The closer the fairness index is to 1, the fairer the offloading is. Furthermore, Fig. 12 presents two distinct flight trajectories of the UAV, with and without considering fairness. It is evident that the DDPG algorithm, when considering fairness, optimizes the UAV's flight trajectory. This allows the UAV to consider all UEs, ensuring each one has the opportunity for task offloading to the UAV for processing.

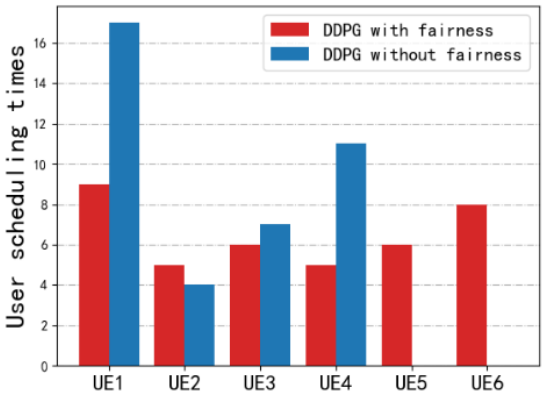


Fig. 10: The influence of offloading fairness on the number of times that the UAV schedule each UE.

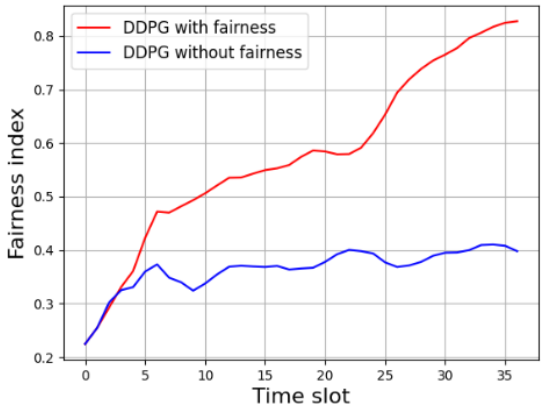


Fig. 11: The trend of fairness index variation within one cycle.

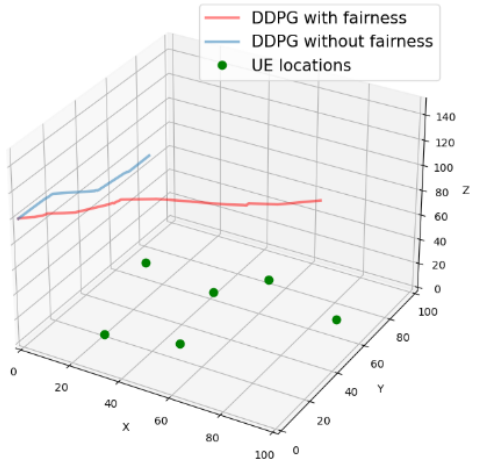


Fig. 12: The influence of offloading fairness on the UAV flight trajectory.

V. Conclusion

In this paper, a UAV-based cloud-edge cooperative task scheduling strategy is proposed for the delay-sensitive and computation-intensive applications running on UEs. We aim at minimizing the sum of the maximum delays over the entire period through collectively optimizing user scheduling, UAV

trajectory, and the task assignment ratio between the cloud and the UAV. Meanwhile, we have introduced a fairness index into the objective function to address the issue of offloading fairness between the UAV and UEs. To address the non-convex optimization problem involving discrete variables, we apply a DDPG-based computation offloading approach to derive the optimal offloading policy. Our simulation experiments reveal that the proposed DDPG outperforms other baseline approaches, particularly with respect to the processing delay. It is also shown that the hybrid offloading outperforms a single cloud offloading or a UAV offloading. Furthermore, we have observed the significance of considering offloading fairness on user scheduling, fairness index, and UAV flight trajectory.

In addition, this work can be expanded into several promising research directions in the future. First, our work assumes that the UAV's altitude remains constant in the vertical direction. However, to be more realistic, we plan to extend the model in future research to optimize the UAV's three-dimensional flight trajectory, including altitude variations, to further improve task offloading efficiency and overall system performance. Second, the current work focuses on using a single UAV to assist the MEC. It could be extended to UAV swarm assisted MEC scenarios, which will require coordinating the motion trajectories of multiple UAVs. Finally, adopting advanced heuristic methods to solve long-term MINLP problems is also a valuable direction for future work, and we plan to introduce heuristic solutions in subsequent studies to further confirm the superiority of the proposed methods.

REFERENCES

[1] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach," *Wireless Network*, vol. 27, no. 4, pp. 2991–3006, 2021.

[2] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628-1656, 2017.

[3] V. Arun and M. Azhagiri, "Design of long-term evolution based mobile edge computing systems to improve 5G systems," in *Proc. International Conference on Edge Computing and Applications (ICECAA)*, Namakkal, India, pp. 160-165, 2023.

[4] B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li, "An online framework for joint network selection and service placement in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 11, pp. 3836-3851, 2022.

[5] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074-54084, 2020.

[6] J. Xiong, P. Guo, Y. Wang, X. Meng, J. Zhang, L. Qian, and Z. Yu, "Multi-agent deep reinforcement learning for task offloading in group distributed manufacturing systems," *Engineering Applications of Artificial Intelligence*, vol. 118, article 105710, 2023.

[7] B. Dai, J. Niu, T. Ren, and M. Atiquzzaman, "Toward mobility-aware computation offloading and resource allocation in end–edge–cloud orchestrated computing," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 19450-19462, 2022.

[8] S. Liu, J. Yin, Z. Zeng, and J. Wu, "Optimal trajectory planning and task assignment for UAV-assisted fog computing," in *Proc. IEEE HPCC/DSS/SmartCity/DependSys*, Hainan, China, pp. 1400-1407, 2022.

[9] K. Xiang and Y. He, "UAV-assisted MEC system considering UAV trajectory and task offloading strategy," in *Proc. International Conference on Communications (ICC)*, Rome, Italy, pp. 4677-4682, 2023.

[10] N. R. Kota and K. Naidu, "Minimizing energy consumption in H-NOMA based UAV-assisted MEC network," *IEEE Communications Letters*, vol. 27, no. 9, pp. 2536-2540, 2023.

[11] Y. Zhang and Z. Mao, "Computation offloading service in UAV-assisted mobile edge computing: a soft actor-critic approach," in *Proc. International Conference on Ubiquitous Communication (Ucom)*, Xi'an, China, pp. 373-378, 2023.

[12] N. Lin, H. Tang, L. Zhao, S. Wan, A. Hawbani, and M. Guizani, "A PDDQNLP algorithm for energy efficient computation offloading in UAV-assisted MEC," *IEEE Transactions on Wireless Communications*, vol. 22, no. 12, pp. 8876-8890, 2023.

[13] H. Guo, Y. Wang, J. Liu, and C. Liu, "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.

[14] H. Guo, X. Zhou, Y. Wang, and J. Liu, "Achieve load balancing in multi-UAV edge computing IoT networks: a dynamic entry and exit mechanism," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 18725-18736, 2022.

[15] H. Guo, X. Zhou, J. Wang, J. Liu, and A. Benslimane, "Intelligent task offloading and resource allocation in digital twin based aerial computing networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 10, pp. 3095-3110, 2023.

[16] R. Alsurdeh, R. N. Calheiros, K. M. Matawie, and B. Javadi, "Hybrid workflow provisioning and scheduling on cooperative edge cloud computing," in *Proc. IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Melbourne, Australia, pp. 445-454, 2021.

[17] X. Wang, X. Li, N. Wang, and X. Qin, "Fine-grained cloud edge collaborative dynamic task scheduling based on DNN layer-partitioning," in *Proc. International Conference on Mobility, Sensing and Networking (MSN)*, Guangzhou, China, pp. 155-162, 2022.

[18] H. Sun, B. Zhang, X. Zhang, Y. Yu, K. Sha, and W. Shi, "FlexEdge: dynamic task scheduling for a UAV-based on-demand mobile edge server," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 15983-16005, 2022.

[19] X. Song, X. Sun, Z. Li, C. Yang, and Y. Wu, "Real-time scheduling strategy for cloud-edge-end collaborative electric vehicles considering energy consumption," in *Proc. Power System and Green Energy Conference (PSGEC)*, Shanghai, China, pp. 644-648, 2023.

[20] J. Li, F. Zhou, W. Li, M. Zhao, X. Yan, Y. Xi, and J. Wu, "Componentized task scheduling in cloud-edge cooperative scenarios based on GNN-enhanced DRL," *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, Miami, FL, USA, pp. 1-4, 2023.

[21] S. Sheng, "A resource collaborative scheduling strategy based on cloud edge framework," in *Proc. IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Chongqing, China, pp. 1740-1744, 2021.

[22] J. Bisht and V. V. Subrahmanyam, "Energy efficient and optimized makespan workflow scheduling algorithm for heterogeneous resources in fog-cloud-edge collaboration," in *Proc. IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, Bhubaneswar, India, pp. 78-83, 2020.

[23] J. Xiong, H. Guo, and J. Liu, "Task offloading in UAV-aided edge computing: bit allocation and trajectory optimization," *IEEE Communications Letters*, vol. 23, no. 3, pp. 538-541, 2019.

[24] X. Diao, J. Zheng, Y. Cai, Y. Wu, and A. Anpalagan, "Fair data allocation and trajectory optimization for UAV-assisted mobile edge computing," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2357-2361, 2019.

[25] M. Coldrey, J. -E. Berg, L. Manholm, C. Larsson, and J. Hansryd, "Non-line-of-sight small cell backhauling using microwave technology," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 78-84, 2013.

[26] L. Ge, P. Dong, H. Zhang, J. -B. Wang, and X. You, "Joint beamforming and trajectory optimization for intelligent reflecting surfaces-assisted UAV communications," *IEEE Access*, vol. 8, pp. 78702-78712, 2020.

[27] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for UAV-enabled mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1879-1892, 2019.

[28] R. Jain, D. M. Chiu, and H. WR, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *CoRR*, vol. cs.NI/9809099, 1998

[29] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: a comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92-108, 2022.

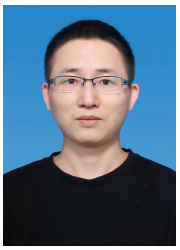
[30] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein, "Policy iteration for decentralized control of Markov decision processes," *Journal of Artificial Intelligence Research*, vol, 34, pp. 89-132, 2009.

[31] Sutton, S. Richard, and G. B. Andrew, "Reinforcement learning: an introduction," *MIT press*, 2018.

[32] M. Luke, J. Ibarz, N. Jaitly, and J. Davidson, "Discrete sequential prediction of continuous actions for deep rl," *arXiv preprint arXiv:1705.05035*, 2017.
[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[35] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. International conference on machine learning*, Pmlr, vol. 32, no. 1, pp. 387-395, 2014.
[36] Z. Zheng, C. Yuan, Z. Lin, Y. Cheng, and H. Wu, "Self-adaptive double bootstrapped DDPG," in *Proc. International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, pp. 3198–3204, 2018.
[37] W. Zhu, X. Chen, L. Jiao, G. Min, and W. Li, "Cost-efficient 6G space-air-ground integrated mobile edge computing for smart city: a PPO-based offloading decision and resource allocation algorithm," in *Proc. IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, Melbourne, Australia, pp. 241-248, 2023.
[38] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1517-1530, 2022.

Haosheng Chen received his B.S. degree in Electronic Information Science and Technology from Shaoguan University, China, in 2023. He is currently pursuing the M.S. degree in Communication Engineering from the School of Electronic Science and Engineering, South China Normal University. His research interests focus primarily on DRL and MEC in wireless communication networks.

Haixia Cui (Senior Member, IEEE) received the M.S. and Ph.D. degrees in Communication Engineering from South China University of Technology (SCUT), Guangzhou, China, in 2005 and 2011, respectively. She is currently a Full Professor with the School of Electronic Science and Engineering (School of Microelectronics), South China Normal University (SCNU), China. From July 2014 to July 2015, she was an Advanced Visiting Scholar (Visiting Associate Professor) with the Department of Electrical and Computer Engineering, the University of British Columbia (UBC), Vancouver, Canada. She has authored or coauthored more than 90 refereed journal and conference papers and 2 books. She also holds about 30 patents. Her current research interests are in the areas of mobile edge computing, vehicular networks, cooperative communication, wireless resource allocation, 5G/6G, multiple access control, and power control in wireless networks.



Jiahuan Wang received the B.S. degree in Mathematics from Southwest Jiaotong University (SWJTU), Chengdu, China, in 2014, and the Ph.D. degree in Information and Communication Engineering from SWJTU, Chengdu, China, in 2022. He is currently an Assistant Professor with the School of Electronic Science and Engineering (School of Microelectronics), South China Normal University, Foshan, China. His research interests include integrated sensing, communication, and waveform design.

Peng Cao received the PhD degree in Communication Engineering from Huazhong University of Science and Technology, Wuhan, China in 2001. His current research interests include the design of short-range wireless communication systems and wearable IoT.

Yejun He (Senior Member, IEEE) received the Ph.D. degree in Information and Communication Engineering from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2005. From 2005 to 2006, he was a Research Associate with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong. From 2006 to 2007, he was a Research Associate with the Department of Electronic Engineering, Faculty of Engineering, The Chinese University of Hong Kong, Hong Kong. In 2012, he joined the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, as a Visiting Professor. From 2013 to 2015, he was an Advanced Visiting Scholar (Visiting Professor) with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. From 2023 to 2024, he is an Advanced Research Scholar (Visiting Professor) with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore.

Dr. He is also an Associate Editor for IEEE Transactions on Antennas and Propagation, IEEE Transactions on Vehicular Technology, IEEE Transactions on Mobile Computing, IEEE Antennas and Propagation Magazine, IEEE Antennas and Wireless Propagation Letters, International Journal of Communication Systems, China Communications, and ZTE Communications. He has served as a Reviewer for various journals, such as IEEE Transactions on Vehicular Technology, IEEE Transactions on Communications, IEEE Transactions on Industrial Electronics, IEEE Transactions on Antennas and Propagation, IEEE Wireless Communications, IEEE Communications Letters, International Journal of Communication Systems, and Wireless Personal Communications.

Mohsen Guizani (Fellow, IEEE) received the BS (with distinction), MS and PhD degrees in Electrical and Computer engineering from Syracuse University, Syracuse, NY, USA in 1985, 1987 and 1990, respectively. He is currently a Professor of Machine Learning at the Mohamed Bin Zayed University of Artificial Intelligence (MBZUAI), Abu Dhabi, UAE. Previously, he worked in different institutions in the USA. His research interests include applied machine learning and artificial intelligence, smart city, Internet of Things (IoT), intelligent autonomous systems, and cybersecurity. He became an IEEE Fellow in 2009 and was listed as a Clarivate Analytics Highly Cited Researcher in Computer Science in 2019, 2020, 2021 and 2022. Dr. Guizani has won several research awards including the "2015 IEEE Communications Society Best Survey Paper Award", the Best ComSoc Journal Paper Award in 2021 as well 5 Best Paper Awards from ICC and Globecom Conferences. He is the author of 11 books, more than 1000 publications and several US patents. He is also the recipient of the 2017 IEEE Communications Society Wireless Technical Committee (WTC) Recognition Award, the 2018 AdHoc Technical Committee Recognition Award, and the 2019 IEEE Communications and Information Security Technical Recognition (CISTC) Award. He served as the Editor-in-Chief of IEEE Network and is currently serving on the Editorial Boards of many IEEE Transactions and Magazines. He was the Chair of the IEEE Communications Society Wireless Technical Committee and the Chair of the TAOS Technical Committee. He served as the IEEE Computer Society Distinguished Speaker and is currently the IEEE ComSoc Distinguished Lecturer.