ISSN 2305-7254 PROCEEDING OF THE 34TH CONFERENCE OF FRUCT ASSOCIATION

Reinforcement Learning with UAV Assistance for Optimized Computation Offloading in Mobile Edge Computing

Aisha Alabsi[1], Ammar Hawbani[1,*], Xingfu Wang[1,*], Saeed Hamood Alsamhi[2], Liang Zhao[3], Ahmed Al-Dubai[4]
[1]University of Science and Technology of China, School of Computer Science and Technology
[2]Insight Centre for Data Analytics, National University of Ireland, Galway, Ireland
[3]Shenyang Aerospace University, School of Computer Science
[4]Computing school of Edinburgh Napier University, United Kingdom
[*]Ammar Hawbani and Xingfu Wang {anmande,wangxfu}@ustc.edu.cn are corresponding authors.

***Abstract*—With the rise of computing-intensive applications like online gaming and telemedicine on user equipment (UE) and the evolution of 5G technology, there is a surge in demand for greater computing resources and power. Yet, UEs have limited resources and batteries. Mobile Cloud Computing (MCC) has emerged as a method to enhance UEs computing capabilities and conserve energy by transferring tasks to the cloud. Mobile Edge Computing (MEC) further aids by reducing delays, although it faces issues like limited resources and unpredictable network conditions. Unmanned Aerial Vehicles (UAVs) offer a remedy by serving as mobile stations for MEC, but optimal offloading decisions in UAV-assisted MEC remain intricate. Addressing this, I propose using Reinforcement Learning (RL), specifically Q-Learning, Deep Q Network (DQN), and Deep Deterministic Policy Gradient (DDPG), to enhance decision-making for offloading. Our focus is on energy efficiency and reduced service delay, and our simulations prove our method's efficacy in UAV-assisted MEC environments.**

I. Introduction

The introduction of 5G wireless communication technology brings about considerable improvements in user equipment (UE), such as higher data rates, reduced latency, and increased capacity compared to earlier generations [1]. 5G technology enables a large number of devices to connect at the same time, creating a more comprehensive Internet of Things (IoT) environment for user equipment (UEs) through its support of massive machine-type communications (mMTC). In 5G's, enhanced mobile broadband (eMBB) capabilities provide users with faster download and upload speeds, improved streaming quality, and virtual reality experiences, thus increasing multimedia consumption, real-time gaming, and video conferencing. Another innovative feature of 5G is network splitting, which allows the creation of custom virtual networks for specific applications or sectors. This feature enables UEs to enjoy optimized network performance, enhanced security, and greater reliability, tailored to their needs. Therefore, 5G technology revolutionizes the potential of UEs, broadening the range of applications and services they can experience with better performance, responsiveness, and adaptability [2].

Mobile Cloud Computing (MCC) combines cloud servers' computing strength with mobile devices' portability, enhancing their performance. MCC allows mobile devices to transfer computational tasks and data storage to the cloud, saving energy, and tackling resource constraints. This gives devices access to vast computing resources, software applications, and on-demand data, broadening their functionality. It also ensures consistent data access across various devices. However, concerns arise regarding data privacy, security, and network dependency in MCC. While MCC increases device capabilities, data security and robust network infrastructure are essential for its effective use [3], [4].

Mobile Edge Computing (MEC) is a modern computational method aimed at reducing delays, minimizing latency, and improving network performance. MEC moves computing resources to the network edge, embedding processing, and storage directly into mobile network edge nodes. This allows UEs to offload computational tasks to nearby edge servers, enhancing system performance and reducing latency. UEs can thus experience reduced transmission delays and greater application responsiveness. Furthermore, MEC leverages the superior resources of edge servers, allowing UEs to undertake complex tasks beyond their device's capacity. This also facilitates data synchronization and sharing across devices, enabling UEs to maintain data consistency and ensure smooth transitions between devices [5], [6].

Unmanned Aerial Vehicles (UAVs) have increasingly been recognized for their ability to improve wireless communications. They can function as aerial base stations, access points, relays, and more, supporting various communication applications like point-to-point and multi-user communications, data collection, secrecy communications, and device-to-device (D2D) communications. This use of UAVs allows wireless networks to expand beyond conventional coverage areas, improving connectivity and accessibility for a broader user base, thus promising significant potential for the future of wireless communication technology [7]. UAVs are becoming more common in various industries because of their size, agility, and cost benefits. They play crucial roles in military tasks such as attack, reconnaissance, and jamming. However, their dynamic use and mobility come with challenges. Given today's complex battle scenarios, there's a growing need for UAVs to operate autonomously. Therefore, the development of decision-making algorithms for UAVs in contemporary aerial combat is a significant research focus, as they require advanced capabilities to navigate and decide in unpredictable conditions [8].

I suggest employing cutting-edge technologies like Reinforcement Learning (RL) to enhance computation-offloading decision-making in UAV-supported MEC. However, the current studies on intelligent maneuver decisions with UAVs through deep reinforcement learning face limitations: (1) Most simulations are predominantly in a three-dimensional (3D) space, focusing on in-depth exploration and analysis. (2) These studies often overlook the impact of radar and weaponry on UAVs, making their application challenging in intricate battlefield settings.

A. *History of Research*

The introduction of UAV technology has brought about significant changes in various industries, offering various applications such as photography, agriculture, surveillance, and disaster response. Integration of UAVs and MEC is a somewhat new trend that started gaining attention in the research field around 2017 and has been growing ever since.

A visual representation of key milestones in the history of the subject is presented in Fig 1.

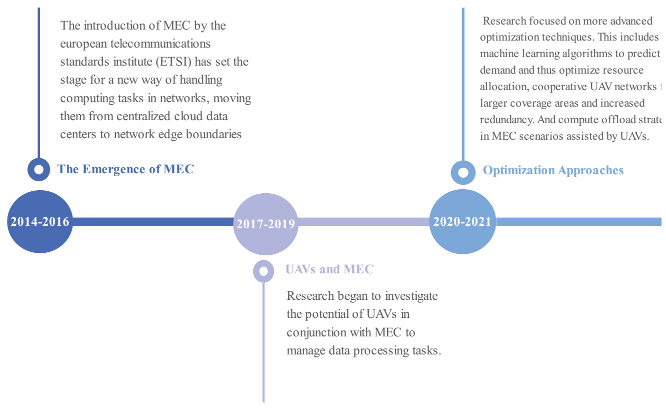
The introduction of MEC by the european telecommunications standards institute (ETSI) has set the stage for a new way of handling computing tasks in networks, moving them from centralized cloud data centers to network edge boundaries
The Emergence of MEC
2014-2016
2017-2019
UAVs and MEC
Research began to investigate the potential of UAVs in conjunction with MEC to manage data processing tasks.
Research focused on more advanced optimization techniques. This includes machine learning algorithms to predict demand and thus optimize resource allocation, cooperative UAV networks larger coverage areas and increased redundancy. And compute offload strat in MEC scenarios assisted by UAVs.
Optimization Approaches
2020-2021

Fig. 1. History of Research

B. *The Research Motivations and Contribution*

Drones, also known as UAVs, have been increasingly in the spotlight due to their promising capabilities in various industries, including MEC. The fusion of UAVs and MEC is an exciting and exciting arena. The proposition entails utilizing the UAV as a portable MEC server to facilitate task processing closer to the ultimate users. This paper introduces the idea of offloading computations, which is the process of transferring intensive computing tasks from devices with limited resources (e.g. mobile phones) to more powerful computing nodes (e.g. UAVs with MEC capabilities).

Here are some key driving forces for optimization in the context of UAV-supported MEC [9] and [10]:

1) **Minimizing delays:** By relocating computation to the network's fringe (through UAVs), we can noticeably reduce the application response time, thus enhancing the user experience.
2) **Preserving battery longevity:** Mobile devices have restricted processing power and battery longevity. Intensive computational activities can quickly deplete these assets. Transferring computation duties to UAVs can help prolong the life of the device's battery.
3) **Improving network efficiency:** By delegating tasks to the network periphery, the duration of data transmission can be minimized. This can result in a decrease in bandwidth consumption, reduced traffic congestion, and improved overall network efficiency.
4) **Enhancing reliability and robustness:** UAVs can be used in scenarios where the ground infrastructure may be compromised or nonexistent, such as areas devastated by disasters, providing reliable and resilient MEC services.
5) **Improving processing capabilities:** Offloading gives devices with limited resources the opportunity to take advantage of the processing power of UAV-MEC servers. This process enables users to operate intricate applications that, without offloading, would either be unfeasible or ineffective on their devices.

In this study, our contributions are as follows.

1) We proposed a MEC model supported by UAVs aimed at enhancing user scheduling, UAV movement, and resource allocation for optimal performance, considering the dynamic nature of channel conditions over time. To tackle the computation offloading challenge, we frame it as a non-convex problem.
2) We present a novel technique based on DDPG that is designed to tackle the difficulties associated with continuous action spaces, thus allowing decision-making in the realm of computation offloading. Additionally, we compare DDPG with another class of algorithms, namely DQN, which is commonly used for environments with discrete action spaces. We are utilizing the DDPG approach to improve the performance and productivity of computation offloading operations.
3) By conducting simulations across a range of system parameters, our goal was to optimize and demonstrate the efficacy of the DDPG algorithm. The results of our simulations revealed that the service delay associated with the DDPG methods we implemented is shorter than that of the DQN algorithms. In these various tests, the DDPG consistently outperformed other DQN algorithms, showcasing its superior performance.

II. Related Work

The computation offloading improvements in MEC systems aided by UAVs are growing rapidly. The goal is to improve the performance and productivity of these systems using UAVs as essential computational tools. A summary of recent relevant research, including a comparative analysis based on crucial criteria, is presented in Table I. Chen et al. [11] introduced a decentralized approach to deep reinforcement learning (DRL) aimed at establishing a coherent policy for dynamic power distribution. Relying on traditional MEC services through static infrastructures presents obstacles, especially in areas with limited communication amenities or in the wake of natural disasters.

Li et al. [12] utilized RL to optimize UEs task migration performance within UAV-assisted MEC systems. Their primary goal was to enhance the efficiency of task transfers. Similarly, Xiong and his team [13] introduced an optimization technique to conserve energy in UAV-based MEC systems. Their strategy focused on optimizing offloading decisions, allocating time, and determining the flight path of UAV's (trajectory) to achieve maximum energy efficiency. The authors of [14] suggested that Device-to-Device (D2D) communication could be used as an additional way to transmit and offload analysis in UAV-MEC systems. Studies on UAV-supported MEC systems have been conducted extensively and have been applied in practice. However, there are still some difficulties that need to be addressed. The capability to assess user equipment and environmental impediments, such as trees or buildings, can have a major effect on system performance, particularly in urban settings. Examining these factors can be essential to optimize system performance [15].

In the study conducted by Cui et al. [16], a multilayer path planning algorithm is proposed, utilizing the RL technique. In contrast to the conventional Q-learning technique, the suggested algorithm collects both global and local data, resulting in a remarkable improvement in total performance. The RL algorithm consists of two layers: The top layer focuses on local data, representing a short-term strategy, while the bottom layer considers global data, functioning as a long-term strategy.

Pham et al. [17] present a framework that employs an RL algorithm for a UAV to locate missing individuals in the aftermath of a natural disaster. To address the challenge of multiple expression structures and achieve faster convergence, they propose a function-approximation-based RL algorithm. The paper [18] emphasizes the importance of deep reinforcement learning (DRL) in optimizing computation offloading in MEC environments. It offers valuable information on the application of RL techniques for task allocation and resource management, showcasing their potential to improve performance and efficiency in mobile edge computing systems. They utilized the DDPG algorithm to discover the most efficient policy for transferring code blocks in diverse settings.

III. Research Methodology

Ultimately, the effectiveness of the suggested approach is gauged. The following section delves into the examination process. Initially, the system architecture of the proposed method is detailed. Then, the recommended algorithms are scrutinized. Finally, the efficiency of the plan is evaluated.

A. System Model

1) Network Model: We analyze a MEC model that includes UAVs, as illustrated in Fig.2. It comprises a set of ground UEs, denoted as $i \in M = \{1, 2, 3, ..., M\}$, a group of UAVs represented by $j \in N = \{1, 2, 3, ..., N\}$, and a collection of edge servers (ESs) located on the ground, with $k \in K = \{1, 2, 3, ..., K\}$. The position of the i-th UEs is indicated as follows: $p_i^T = (x_i, y_i, \theta)$, where $i \in M$, and the location of the k-th ES is denoted as $p_k^F = (x_k, y_k, \theta)$, where $k \in K$.

During each time interval, the UAV remains stationary and establishes a connection with a single UEs for the purpose of transmission. The UAV functions at a consistent altitude H. The location coordinates of the UAV at position j can be denoted as $p_j^U = (x_j, y_j, H)$, where j is a member of the set N. We divide the total time T into smaller intervals of Δ time in order to facilitate system operations. Δ is chosen large enough to ensure that the positions of the UEs and UAVs stay nearly fixed within a single time period.

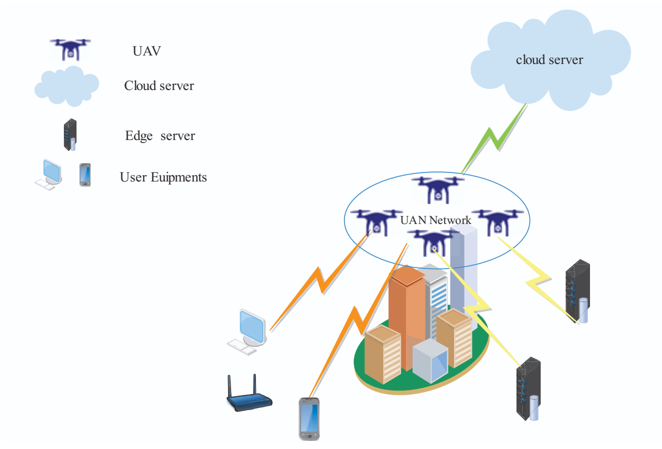

Fig. 2. Network Model

B. The RL algorithms

To address the challenges of determining efficient computing offloading policies and selecting the optimal transmission path for tasks relayed through multiple UAVs, RL methods are proposed. Agents in RL interact with their environment, continually improving their decision-making skills through repeated learning from experience and testing. This study conducts a comprehensive analysis of five fundamental components that are integral to the process of RL: namely, the agent, environment, state representation, action selection, and reward function. To meet our research goals, we will utilize two renowned RL algorithms: DQN and DDPG. Both algorithms have demonstrated outstanding performance in numerous RL tasks, making them well-suited for our research objectives.

1) DQN: This algorithm is frequently employed in deep reinforcement learning. It merges deep learning and Q-learning to tackle intricate RL problems characterized by high-dimensional state spaces. DQN demonstrates exceptional performance, especially in situations characterized by extensive and continuous state and action spaces. Fundamentally, DQN utilizes a deep neural network, often a convolutional neural network (CNN), to approximate the Q-values associated with various state-action pairs.

TABLE I. BRIEF SUMMARY ON THE REVIEWED STATE-OF-THE-ART LITERATURES

Ref	RL Approach	Research Problem	Objective
[19]	Distributed-DQN	Optimize offloading strategy	Maximise the amount of computing tasks.
[20]	DDPG	Improve user scheduling, adjust the proportion of task offloading and optimize the flight angle and speed of the UAV	Minimize the maximum processing delay.
[21]	Deep Learning	Optimal offloading decision	Attempt to reduce the amount of time and energy expended.
[22]	DDPG	Offloading decision optimization	Ensure equitable treatment of all users.
Our research	DQN and DDPG	Offloading decision optimization	Minimizing the maximum processing delay.

The Q-learning update equation used in DQN is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)+\alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (1)$$

The temporal difference error is calculated by subtracting the updated Q-value from the original Q-value in each step. The learning rate is applied to this difference and the result is added to the original Q-value to give the updated Q-value. The process is repeated over many episodes to gradually improve the policy and value estimates. The aim of the DQN algorithm is to iteratively update the Q-values, striving for convergence towards the optimal Q-values that maximize the total discounted rewards. The Q-learning update equation ensures that the Q-values are adjusted according to the current reward and the estimated future rewards. The Algorithm (1) describes the DQN training process.

2) DDPG: This RL algorithm combines the advantages of value-based and policy-based approaches. This software was specifically created to address continuous action space issues, making it ideal for tasks that involve continuous and high-dimensional actions. DDPG relies on two primary elements: the actor-network and the critic network. The actor-network formulates a policy associating states with actions, enabling the agent to decide according to the observed state.

The DDPG algorithm uses two distinct Deep Neural Networks (DNNs) to simulate the actor-network $l(s|\theta^\mu)$ and the critic network $Q(s, a|\theta^Q)$. The actor network is responsible for the policy implementation, while the critic network is responsible for the Q-value calculation. Furthermore, the two networks include target networks of the same design: the actor target network, denoted as l_0, utilizes parameter h_{l_0}, while the critic target network, represented as Q_0, employs parameter h_{Q_0}. The formulations for these networks are provided below:

1) **Actor-network:**

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i,a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i} \quad (2)$$

2) **Critic network:**

$$L(\theta^Q)=\frac{1}{N} \sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2 \quad (3)$$

Algorithm 1 Deep Q-Network (DQN)

```
1: Start with a replay memory $D$ set to a maximum limit of $N$.
2: Initialize the action-value function $Q$ with random weights $\theta$.
3: Establish the target action-value function $\tilde{Q}$ with weights $\theta^- = \theta$.
4: for each episode $m$ ranging from 1 to $M$ do
5:     Define the sequence $s_1$ as $s_1 = \{x_1\}$ and preprocess to get $\psi_1$ where $\psi_1 = \psi(s_1)$.
6:     for each time step $t$ from 1 to $T$ do
7:         if a randomly generated value $\leq \epsilon$ then
8:             Choose a random action $a_t$.
9:         else
10:            Select the action $a_t$ that maximizes $Q(\psi(s_t), a; \theta)$.
11:        end if
12:        Execute action $a_t$, receive reward $r_t$, and obtain the image $x_{t+1}$.
13:        Using inputs $s_t$, $a_t$, and $x_{t+1}$, modify the state to $s_{t+1}$ and preprocess to determine $\psi_{t+1} = \psi(s_{t+1})$.
14:        Record the transition sequence $(\psi_t, a_t, r_t, \psi_{t+1})$ into $D$.
15:        Draw a random subset of transition sequences $(\psi_j, a_j, r_j, \psi_{j+1})$ from $D$.
16:        if $\psi_{j+1}$ indicates a termination then
17:            Assign the target value $y_j$ to be $r_j$.
18:        else
19:            Calculate the target value $y_j$ using the formula $r_j + \gamma \max_{a'} \tilde{Q}(\psi_{j+1}, a'; \theta^-)$.
20:        end if
21:        Apply a gradient descent adjustment on the equation $(y_j - Q(\psi_j, a_j; \theta))^2$, considering the weight $\theta$.
22:        if the time step $t$ is a multiple of $C$ then
23:            Update the target network $\tilde{Q}$ to match $Q$.
24:        end if
25:    end for
26: end for=0
```

The algorithm is designed to execute multiple episodes, each of which involves interactions with the environment and the collection of experience tuples, which include the state, action, reward, and subsequent state. The DDPG algorithm uses neural networks that are trained with experience tuples to create its actor and critic components. By taking advantage of this training procedure, the actor network can be trained to imitate the ideal policy, while the critic network can be trained to imitate the related Q-values.

Each experience tuple consists of four elements:

A. **State space**

The state of the system can be determined by K UEs, a UAV, and influences, it is represented as:

$$\begin{aligned} s_i = (&B_{\text{energy}}(i), m_1(i), m_2(i), \ldots, m_k(i), \\ &n_1(i), n_2(i), \ldots, n_L(i), R_{\text{remain}}(i), \\ &R_1(i), R_2(i), \ldots, R_L(i), g_1(i), g_2(i), \ldots, g_L(i)) \end{aligned} \tag{4}$$

Given: $B_{\text{energy}}(i)$ signifies the remaining energy in the UAV's battery during the i-th time interval, $m_k(i)$ provides the positional details of the UAV, $n_L(i)$ corresponds to the location data of the UEs accessed by the UAVs, $R_{\text{remain}}(i)$ stands for the cumulative tasks the system must finish within the designated duration, The notation $R_L(i)$ signifies the scale of the task spontaneously produced by the UEs during the i-th period, and $g_L(i)$ determines if the UE's signal faces obstructions due to any blockages.

B. **Action space**

Considering the current state of the system and the environment that has been observed, the behavior of the system can be described in the following manner.

Given the current status of the system and insights from the environment, the system's behavior can be articulated as:

$$a_j = (u(i) + \alpha(i) + w(i) + T_u(i)) \tag{5}$$

Where $u(i)$ signifies the UEs to be served and $u(i) \in [0, u]$, $\alpha(i)$ symbolizes the flight angle of the UAV and $\alpha(i) \in [0, 2\pi]$, $w(i)$ represents the flight speed of the UAV and $w(i) \in [0, w_{\text{max}}]$, and $T_u(i)$ denotes the task offloading ratio and $T_u(i) \in [0, 1]$.

C. **Reward**

The performance of the DDPG framework is significantly influenced by the rewards an agent obtains. Creating an appropriate reward system is essential to direct the agent's actions. Our research primarily aims to minimize information processing time, thereby maximizing the agent's reward. The r_i is designated as the reward function and is expressed as follows:

$$r_i = R(s_i, a_i) = -\Delta_{\text{delay}}(i) \tag{6}$$

Where $\Delta_{\text{delay}}(i)$ symbolizes the processing delay at step 'i':

$$\Delta_{\text{delay}}(i) = \sum_{j=1}^{Y} a_j(i) \max\{f_{\text{local},j}(i), t_{\text{UAV},j}(i) + t_{\text{tr},j}(i)\} \tag{7}$$

And, for $a_j(i)$, we define it as:

$$a_j(i) = \begin{cases} 1 & \text{if } j = j_0 \\ 0 & \text{otherwise} \end{cases}$$

Where, R represents the reward, Δ_{delay} is the processing delay, and j is the new index running from 1 to Y, j_0 is the specific index where $a_j(i)$ is 1.

D. **State normalization**

The state normalization algorithm is used to adjust and standardize state variables within a machine learning system. We recommend this algorithm as an initial step to minimize the impact of varying magnitudes on the input data. In our research, we apply the state normalization algorithm to fine-tune the variables we use.

The state normalization algorithm has five scaling factors, which are represented by γ_b, γ_x, γ_y, γ_{rm}, and γ_{UE}.

The following expression denotes each scaling factor:

-γ_b: Using this method decreases the battery capacity of the UAV.

-γ_x, γ_y: The objective is to minimize the coordinates of UEs and UAV x and y.

-γ_{rm}: The objective is to reduce the total number of tasks that remain to be completed throughout the duration.

-γ_{UE}: The objective is to minimize the amount of time each user equipment task requires in the i-th time slot.

E. **DDPG algorithm training and testing**

This algorithm is widely used in reinforcement learning to train continuous action spaces, which are a combination of deep learning and policy gradient techniques. This algorithm simultaneously learns a Q function and a policy. The core idea of DDPG is to use a single neural network to directly determine the optimal policy and a second network to calculate the expected value of taking a certain action in a given state.

The DDPG algorithm employs an actor-critic model where the actor proposes actions and the critic assesses their value. During its training process, DDPG leverages experience replay to enhance learning and implements soft updates for maintaining stability. For exploration within continuous action spaces, it uses noise, notably from the Ornstein-Uhlenbeck process. Additionally, training stability is augmented by normalizing both features and rewards.

As it shifts to the testing phase, actions are solely dictated by the actor network, eschewing the exploratory noise prominent in training for more deterministic outcomes. The algorithm's effectiveness is measured through metrics like average reward and success ratio, complemented by visual analysis to decode the agent's decisions. Both the actor and critic networks are trained using randomly selected experiences.

To gauge the effectiveness of the DDPG employed in the computation offloading technique, we adopt a two-phase approach: training followed by testing. As detailed in Algorithm (2), the training phase comprises continuous adjustments to the parameters of both the critic and actor networks of the behavior policy. These adjustments aim to optimize the process of computation transfer. During the testing stage, the previously trained actor-network, denoted as θ^{μ}, is employed to execute computation offloading in a real-world setting. The steps of the procedure are outlined in Algorithm(3).

Algorithm 2 DDPG Training Algorithm

```
1: **Input**: Episode duration $E$, Sample size $I$, Learning rate for Critic $\alpha_{\text{critic}}$, Learning rate for Actor $\alpha_{\text{actor}}$, Discount factor $\gamma$, Replay memory buffer $B_m$, Batch dimension $N$, Gaussian noise $n$ with mean $\mu_e = n_0$, and uniform standard deviation $\sigma_{e,i} = \sigma_e$.
2: Initialize actor network weights $\theta^{\mu}$ and critic network weights $\theta^{Q}$ randomly.
3: Set target network weights: $\theta^{\mu} \leftarrow \theta^{\mu'}$ and $\theta^{Q} \leftarrow \theta^{Q'}$.
4: Clear the replay buffer $B_m$.
5: **for** episode $e = 1$ to $E$ **do**
6:     Reset UAV-assisted MEC model simulation parameters and get initial state $s_1$.
7:     **for** $i = 1$ through $I$ **do**
8:         Convert state $s_i$ to normalized form $\hat{s}_i$.
9:         Calculate action $a$ using the actor network $\theta^{\mu}$ and the noise component $n_i$ as:
           $a = \text{clamp}(\theta^{\mu}(\hat{s}_i, \theta^{\mu}) + n_i, -1, 1)$.
10:        Implement action $a_i$, attain reward $r_i$, and note the resulting state $s_{i+1}$.
11:        Convert the following state $s_{i+1}$ to its normalized version $\hat{s}_{i+1}$.
12:        **if** storage buffer $B_m$ has available space **then**
13:            Archive the tuple $(\hat{s}_i, a_i, r_i, \hat{s}_{i+1})$ within $B_m$.
14:        **else**
15:            Overwrite a random tuple in $B_m$ using $(\hat{s}_i, a_i, r_i, \hat{s}_{i+1})$.
16:            Draw a mini-batch of $I$ random tuples $(\hat{s}_j, a_j, r_j, \hat{s}_{j+1})$ for each $j$ from 1 to $I$ in $B_m$.
17:            Calculate $y_j$ as $r_j + \gamma Q'(\hat{s}_{j+1}, \mu'(\hat{s}_{j+1}|\theta^{\mu'}), \theta^{Q'})$.
18:            Update critic network's weights $\theta^{Q}$ to minimize the loss.
19:            Enhance actor network's weights $\theta^{\mu}$ using the determined policy gradient.
20:        **end if**
21:    **end for**
22: **end for**
23: **return** Actor-network $\mu(\hat{s}|\theta^{\mu})$. =0
```

Algorithm 3 DDPG Testing Algorithm

```
1: **Input:**
2: Testing episode length $E'$
3: Testing step length $N$
4: Trained actor network $\mu(\hat{s}|\theta^{\mu})$
5: **Initial states:**
       $B_{\text{energy}}(i), m_1(i), m_2(i), \ldots, m_k(i),$
       $n_1(i), n_2(i), \ldots, n_L(i),$
       $R_{\text{remain}}(i), R_1(i), R_2(i), \ldots, R_L(i),$
       $g_1(i), g_2(i), \ldots, g_L(i)$
6: Normalization parameters: $\gamma_b, \gamma_x, \gamma_y, \gamma_{Drm}, \gamma_{DUE}$
7: **Output:** Reward $r_i$
8: **for** episode $e$ from 1 to $E'$ **do**
9:     STATE Initialize state $s_i$ by resetting UAV-assisted MEC model parameters
10:    **for** step $i$ from 1 to $I$ **do**
11:        Compute normalized state $\hat{s}_i$ from $s_i$
12:        Determine action using actor network: $a_i = \mu(\hat{s}_i|\theta^{\mu})$
13:        Execute action $a_i$ to receive reward $r_i$
14:    **end for**
15: **end for**=0
```

IV. Simulation results and discussions

A. Parametric analysis

First, we conduct a series of experiments to determine the optimal values of the important hyperparameters used in the algorithm comparisons. The convergence outcomes of the recommended algorithms (DQN, DDPG) vary based on diverse parameters such as control exploration, discount factors, and learning rate, as detailed below:

1) The control exploration: The exploration parameter significantly influences how well the DDPG algorithm converges shown in Fig. 3.

It demonstrates the performance differences of the DDPG algorithm concerning processing delay when various exploration parameters σ are applied. The figure illustrates the control exploration for DDPG across various episodes, evaluated using different exploration parameter values (σ). As the number of episodes increases, the rewards for different values of σ appear to converge, with the curve for σ=0.05 showcasing a steeper ascent compared to the others before plateauing. By observing the trajectories and the reward values, the exploration parameter σ=0.05 seems to achieve the highest reward in the fewest episodes, indicating it may be the most optimal value for exploration among the presented parameters.

2) The discount factors: Fig. 4 illustrates the impact of different discount factors (γ) on the DDPG algorithm's performance across various episodes. As the discount factor rises, there's a noticeable improvement in the rewards obtained. For lower values like γ=0.1 and γ=0.001, the performance is notably weaker, indicating that these settings place minimal emphasis on future rewards, potentially leading to short-sighted decision-making. The curve for γ=0.5 shows moderate

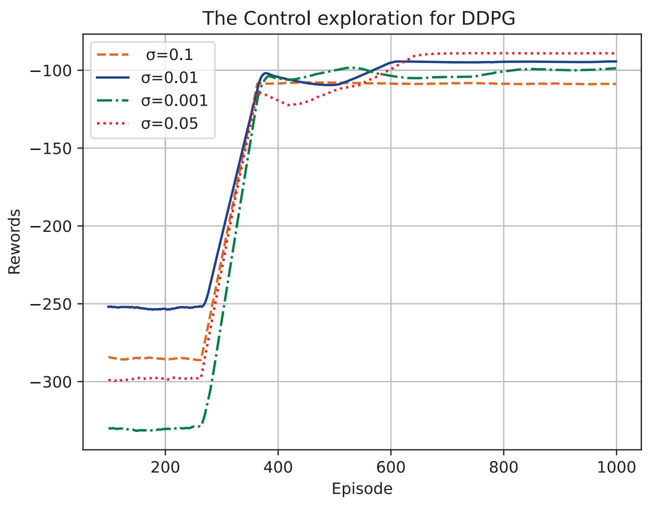

Fig. 3. Performance variation of the rewards across different settings of the exploration parameter

performance, suggesting a more balanced consideration between immediate and future rewards. Meanwhile, γ=0.9 offers better results, emphasizing the significance of longer-term rewards without completely maximizing them. In comparison to these, γ=0.999 offers the highest rewards, but the nuances in performance across different discount factors underscore the importance of fine-tuning this parameter based on specific problem contexts and desired outcomes. Thus, γ=0.999 is the optimal discount factor here.

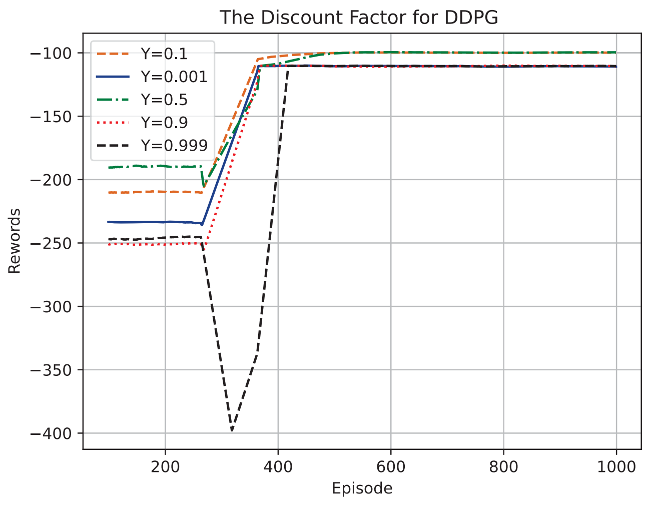

Fig. 4. Performance analysis of the DDPG algorithm's convergence with varying discount factors

In Fig. 5, the performance of the DQN algorithm is influenced by various discount factors (γ) across a series of episodes. On the y-axis, we observe the rewards, and the x-axis represents the episodes. The curve for γ=0.1 remains largely below other curves, suggesting a weaker convergence when prioritizing immediate rewards. The γ=0.01 and γ=0.001 trajectories depict more fluctuations, indicating potential instability or sensitivity to certain episodes. Interestingly, γ=0.5 showcases an initial dip in rewards before stabilizing, while γ=0.9 exhibits significant volatility, highlighting the unpredictable nature of certain discount factors. Notably, while most discount factors exhibit fluctuating performances, the γ=0.99 curve demonstrates a relatively smoother and higher reward trajectory, especially toward the later episodes. This suggests a superior balance between short-term and long-term rewards for this discount factor. In this context, γ=0.99 appears to be the optimal discount factor for the DQN algorithm.

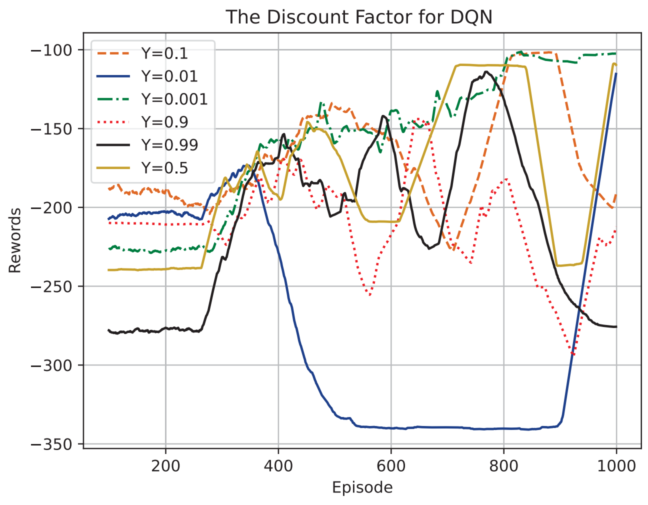

Fig. 5. Performance analysis of the DQN algorithm's convergence with varying discount factors

3) The learning rate: Fig. 6 delineates the performance of the DDPG algorithm at varying learning rates for both the actor and critic networks over a series of episodes. Observably, the pair with $\alpha_{\text{actor}} = 0.01$ and $\alpha_{\text{critic}} = 0.02$ showcases a superior and more stable reward trajectory compared to other combinations. On the other hand, extremely low learning rates like $\alpha_{\text{actor}} = 0.0001$ and $\alpha_{\text{critic}} = 0.0002$ result in the least rewarding outcomes, suggesting an inability to adapt quickly to the environment. In this context, the optimal learning rate combination for the DDPG algorithm appears to be $\alpha_{\text{actor}} = 0.01$ and $\alpha_{\text{critic}} = 0.02$.

In Fig. 7, the performance trajectory of the DQN algorithm for different learning rates over numerous episodes. Evidently, a learning rate of $\alpha = 0.01$ offers the most consistent and optimal reward curve, demonstrating stability and promising results. In contrast, the highest learning rate $\alpha = 0.1$ seems to oscillate substantially, indicating potential overfitting or erratic learning. The extremely low learning rates, $\alpha = 0.001$ and $\alpha = 0.0001$, present less favorable outcomes, suggesting they might be too conservative for rapid

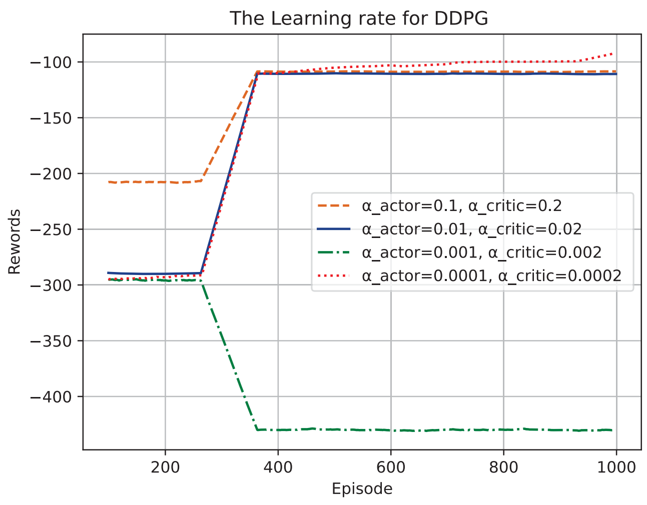

Fig. 6. Performance analysis of the DDPG algorithm's convergence with varying learning rate

environment adaptation. Therefore, $\alpha = 0.01$ emerges as the most suitable learning rate for the DQN algorithm based on the depicted results.

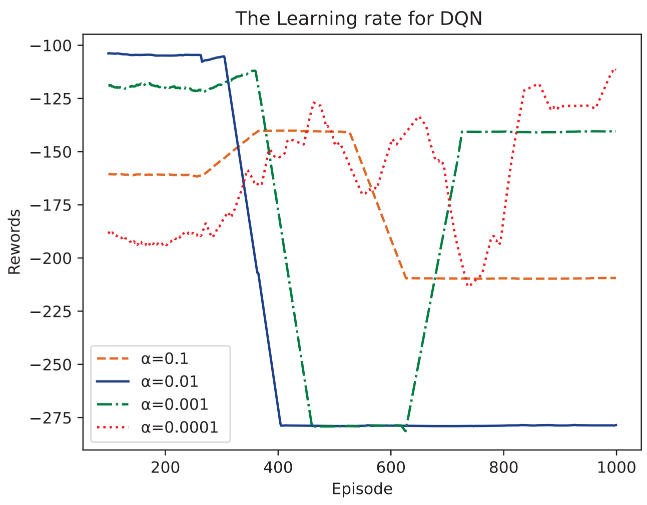

Fig. 7. Performance analysis of the DQN algorithm's convergence with varying learning rate

B. Performance comparison

We analyze a two-dimensional (2D) square region in the MEC model assisted by UAVs, with the simulation parameters listed in Table (II). The two-dimensional area is the platform on which the system operates, and the simulation parameters provide the details and configurations required to carry out the simulations.

Three methods of utilizing this model are outlined below:

- The Offload-only approach: In this model, the UEs hand over all their computational duties to the UAV for a certain period of time. A UAV that is positioned at a fixed location in the region can provide computational assistance to UEs by sending the tasks to the MEC server that is located within it.
- The offloading algorithm that employs DQN is compared to DDPG. DQN functions in a conventional discrete action space, whereas DDPG utilizes a continuous action space.
- The algorithm evaluates the performance of computation offload using DDPG. This approach employs a continuous action space to tackle the intricacies of computation offloading.

We investigated the performance disparities between the value approximation and the policy approximation in RL algorithms, specifically in the sequential cases of DQN and DDPG, considering factors such as delay, UEs, and the offloading ratio. The observation made, based on Figure (8), that DDPG showed a lesser delay than DQN in the episodes examined is consistent with the structural differences between the two algorithms. This doesn't mean DDPG is always superior to DQN, but in the specific context of the study and in environments where the continuous nature of actions is crucial, DDPG might offer advantages in processing speed.

However, when deciding between these algorithms in practical applications, it's essential to consider other factors like stability of learning, ease of implementation, and overall performance, not just processing delay.

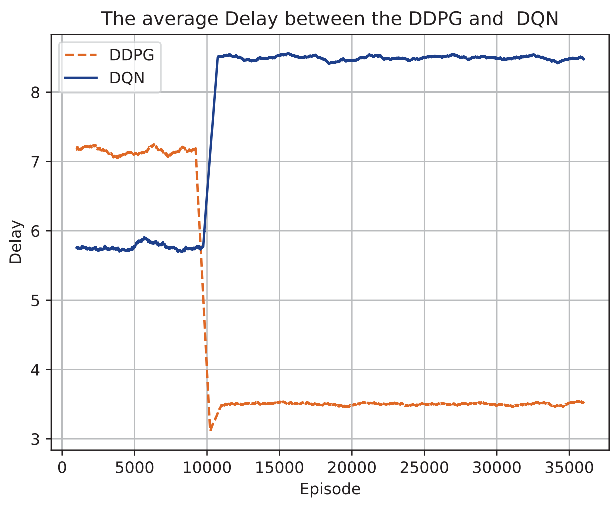

Fig. 8. Performance analysis of delay under various RL algorithms

In Fig. 9, this figure visually contrasts the performance of two algorithms, DDPG and DQN, in relation to the average processing delay as the number of UEs changes. The choice between the two would depend on the specific s cenario and the number of UEs involved. The comparison of average processing delay for varying numbers of UEs (from 0 to 0.5)

TABLE II. DEFAULT SIMULATION PARAMETERS

Par	Value	Description	Par	Value	Description
K	4	Num Of UEs	S	1000 c/bit	CPU Cycle per bit
L, W, H	100 m	UAV(Hight, width)	M_{UAV}	9.65 kg	UAV gross mass
T	400 s	Whole time period	I	40	Time slots
v_{max}	50 m/s	Maximum UAV flight speed	t_{fly}	1 s	Flight time of UAV
α_0	- 50 dB	Channel power gain	B	1 MHz	Transmission bandwidth
σ^2	- 100 dBm	Noise power of the receiver	P_{NLOS}	20 dB	Penetration loss
P_{up}	0.1 W	Transmission power of UEs	E_b	500 kJ	UAV battery capacity
f_{UE}	0.6 GHz	Computing capability of the UEs server	f_{UAV}	1.2 GHz	Computing capability of the MEC server

highlights that the DDPG algorithm seems to have a higher processing delay compared to the DQN algorithm for a specific user equipment set.

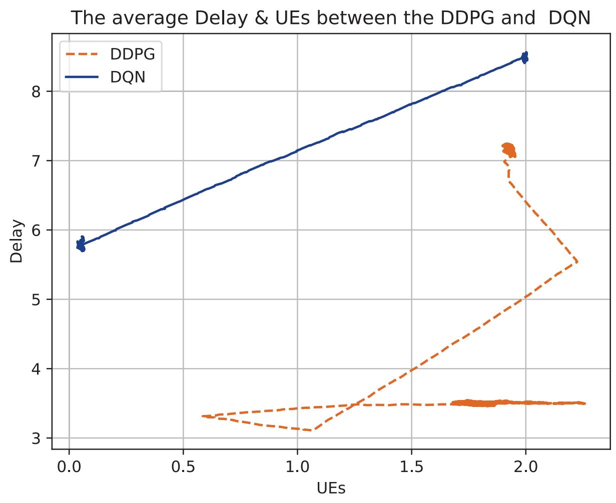

Fig. 9 Performance of DQN and DDPG under different UEs

In Fig. 10, the offloading ratio for DDPG is moderate. However, as training progresses, the algorithm seems to prefer offloading almost all tasks/data (as seen by the ratio approaching 1.0). This suggests that with more experience or training, DDPG finds it more beneficial to offload. However, the DQN starts with a higher offloading ratio compared to its eventual performance but rapidly reduces its offloading preference. This indicates that, over time, DQN may find local processing more efficient than offloading. The contrasting behavior between DDPG and DQN is evident. While DDPG tends to offload more as it gains more experience, DQN does the opposite, reducing its offloading preference.

The DDPG algorithm was found to have the least amount of delay compared to the other schemes. Its capacity to enhance continuous activities and determine an ideal control strategy can be credited to it.

V. CONCLUSION

In light of the growing demands of computing-intensive applications on UEs and the challenges posed by limited resources, solutions such as MCC and MEC have been explored. Though MEC presents its set of challenges, the introduction of UAVs offers a promising solution. Our proposal to employ Reinforcement Learning techniques, specifically Q-Learning, DQN, and DDPG, has been shown to effectively optimize computation offloading decisions in UAV-assisted MEC scenarios. Simulations validate our approach's superiority in enhancing system performance in terms of energy efficiency and minimizing service delay compared to other existing methodologies. The simulation data suggests that the DDPG algorithm surpasses the DQN algorithm concerning processing delay. In subsequent studies, we intend to examine the efficacy of our algorithm in conjunction with other reinforcement learning policy approximation methods like TRPO and PRO2.

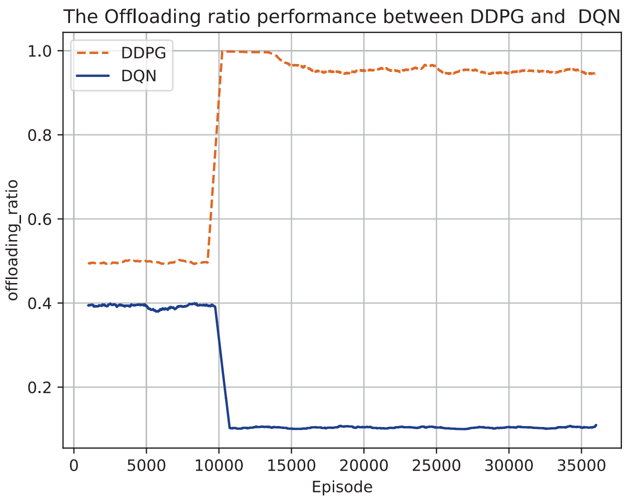

Fig. 10. Offloading ratio performance of DQN and DDPG

ACKNOWLEDGMENT

This study received support from the Innovation Team and Talent Cultivation Program overseen by the National Administration of Traditional Chinese Medicine (Reference No. ZYYCXTD-D-202208). The research related to Saeed was funded by Science Foundation Ireland, with the grant identifier SFI/12/RC/2289_P2. Al-Dubai extends gratitude for the assistance provided by the UK Engineering and Physical

Sciences Research Council (EPSRC) through the program grant titled COG-MHEAR (Grant ID: EP/T024917/1).

REFERENCES

[1] D. M. West, "How 5g technology enables the health internet of things," *Brookings Center for Technology Innovation*, vol. 3, no. 1, p. 20, 2016.

[2] W. S. H. M. W. Ahmad, N. A. M. Radzi, F. Samidi, A. Ismail, F. Abdullah, M. Z. Jamaludin, and M. Zakaria, "5g technology: Towards dynamic spectrum sharing using cognitive radio networks," *IEEE access*, vol. 8, pp. 14 460–14 488, 2020.

[3] S. V. Shinde and S. Sonavane, "An electronic tattoo based wireless body area network (et-wban) to conform ieee 802.15. 6: A review and proposal," in *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*. IEEE, 2018, pp. 269–274.

[4] S. Shamshirband, M. Fathi, A. T. Chronopoulos, A. Montieri, F. Palumbo, and A. Pescapè, "Computational intelligence intrusion detection techniques in mobile cloud computing environments: Review, taxonomy, and open research issues," *Journal of Information Security and Applications*, vol. 55, p. 102582, 2020.

[5] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 1965–1978, 2018.

[6] E. Ahmed and M. H. Rehmani, "Mobile edge computing: opportunities, solutions, and challenges," pp. 59–63, 2017.

[7] L. Xie, X. Cao, J. Xu, and R. Zhang, "Uav-enabled wireless power transfer: A tutorial overview," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 2042–2064, 2021.

[8] M. S. Alam and J. Oluoch, "A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (uavs)," *Expert Systems with Applications*, vol. 179, p. 115091, 2021.

[9] Y. Xu, T. Zhang, D. Yang, Y. Liu, and M. Tao, "Joint resource and trajectory optimization for security in uav-assisted mec systems," *IEEE Transactions on Communications*, vol. 69, no. 1, pp. 573–588, 2020.

[10] M. Qin, N. Cheng, Z. Jing, T. Yang, W. Xu, Q. Yang, and R. R. Rao, "Service-oriented energy-latency tradeoff for iot task partial offloading in mec-enhanced multi-rat networks," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1896–1907, 2020.

[11] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, no. 1, pp. 1–21, 2020.

[12] J. Li, Q. Liu, P. Wu, F. Shu, and S. Jin, "Task offloading for uav-based mobile edge computing via deep reinforcement learning," in *2018 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2018, pp. 798–802.

[13] 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.

[14] M. M. Selim, M. Rihan, Y. Yang, and J. Ma, "Optimal task partitioning, bit allocation and trajectory for d2d-assisted uav-mec systems," *Peer-to-Peer Networking and Applications*, vol. 14, pp. 215–224, 2021.

[15] 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. 78 702–78 712, 2020.

[16] Z. Cui and Y. Wang, "Uav path planning based on multi-layer reinforcement learning technique," *IEEE Access*, vol. 9, pp. 59 486–59 497, 2021.

[17] H. X. Pham, H. M. La, D. Feil-Seifer, and L. Van Nguyen, "Reinforcement learning for autonomous uav navigation using function approximation," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2018, pp. 1–6.

[18] M. Chen, T. Wang, S. Zhang, and A. Liu, "Deep reinforcement learning for computation offloading in mobile edge computing environment," *Computer Communications*, vol. 175, pp. 1–12, 2021.

[19] X. Zhang and Y. Wang, "Deepmecagent: Multi-agent computing resource allocation for uav-assisted mobile edge computing in distributed iot system," *Applied Intelligence*, vol. 53, no. 1, pp. 1180–1191, 2023.

[20] 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 Networks*, vol. 27, no. 4, pp. 2991–3006, 2021.

[21] M. Mukherjee, V. Kumar, A. Lat, M. Guo, R. Matam, and Y. Lv, "Distributed deep learning-based task offloading for uav-enabled mobile edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1208–1212.

[22] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and L. Hanzo, "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, 2020.