Computation Offloading via Multi-Agent Deep Reinforcement Learning in Aerial Hierarchical Edge Computing Systems

Yuanyuan Wang, Chi Zhang, *Member, IEEE,* Taiheng Ge, and Miao Pan, *Member, IEEE*

Abstract—The exponential growth of Internet of Things (IoT) devices and emerging applications have significantly increased the requirements for ubiquitous connectivity and efficient computing paradigms. Traditional terrestrial edge computing architectures cannot provide massive IoT connectivity worldwide. In this paper, we propose an aerial hierarchical mobile edge computing system composed of high-altitude platforms (HAPs) and unmanned aerial vehicles (UAVs). In particular, we consider non-divisible tasks and formulate a task offloading problem to minimize the long-term processing cost of tasks while satisfying the queueing mechanism in the offloading procedure and processing procedure of tasks. We propose a multi-agent deep reinforcement learning (DRL) based computation offloading algorithm in which each device can make its offloading decision according to local observations. Due to the limited computing resources of UAVs, high task loads of UAVs will increase the ratio of abandoning offloaded tasks. To increase the success ratio of completing tasks, the convolutional LSTM (ConvLSTM) network is utilized to estimate the future task loads of UAVs. In addition, a prioritized experience replay (PER) method is proposed to increase the convergence speed and improve the training stability. The experimental results demonstrate that the proposed computation offloading algorithm outperforms other benchmark methods.

Index Terms—Aerial computing, mobile edge computing, deep reinforcement learning, computation offloading.

1 Introduction

With the development of sixth-generation wireless communication systems (6Gs), several widely anticipated applications, such as smart farming, intelligent transportation, and depopulated zone detection, can be realized through the deployment of large-scale Internet-of-Thing (IoT) networks [1]. Considering that most IoT devices are low-cost devices equipped with limited storage and computing resources, these devices cannot execute computation-intensive tasks locally [2]. Fortunately, the mobile edge computing (MEC) paradigm has been proposed to assist IoT devices in executing computing tasks [3]. However, traditional terrestrial-network-based edge computing systems are inadequate for IoT devices in emergency scenarios and remote areas. Hence, it is essential to design a novel effective MEC system for such circumstances.

As supplements to terrestrial networks, unmanned aerial vehicles (UAVs), high-altitude platforms (HAPs), and satellites with different orbits can provide effective coverage to areas where terrestrial infrastructures are unavailable or overloaded [4]. Compared with satellites, HAPs can stay at a quasi-stationary position at an altitude of approximately 20 km for several months to provide stable, low-latency, and affordable services for IoT devices [5]. HAPs can open the way to new critical IoT services. For example, an HAP can be equipped with various kinds of sensors to monitor the environment and caching servers to record the events occurring within the HAP zone. Many existing works have been devoted to studying the usage of HAPs and UAVs in several applications and projects [6], [7], [8]. For example, the HAWK30 project executed by HAPSMobile aims to deploy multiple HAPs to connect mobile users, UAVs, and IoT devices around the world [9]. As we know, in Puerto Rico's 2017 disaster, Loon technology was a cost-effective coverage solution to the difficult challenge of providing Internet access and communication services to people for post-disaster recovery [10]. High-altitude balloons (HABs) equipped with sensors, caching servers, and computing servers can detect and predict important events deduced from sensor readings in real time. These functionalities can assist emergency and disaster teams in building new models for natural disasters based on sensor readings, which can help forecast future events more accurately. UAVs are well known due to their low transmission delay and flexibility of deployment [11]. Devices with limited power supplies cannot connect directly to HAPs, so they can seek the help of UAVs. However, the computing and storage resources and endurance time of a UAV are much shorter than those of an HAP. Hence, UAVs can only execute a limited number of computation-intensive tasks for IoT devices. Unlike UAVs with limited onboard energy and resources, HAPs have much longer endurance time and are capable of executing large quantities of computation-intensive tasks due to their

- *Yuanyuan Wang and Chi Zhang are with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230027, China.*
  E-mail: yywang95@mail.ustc.edu.cn; chizhang@ustc.edu.cn
- *Taiheng Ge is with the School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, China.*
  E-mail: gth@mail.ustc.edu.cn
- *Miao Pan is with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204, USA.*
  E-mail: mpan2@uh.edu
- *Chi Zhang is the corresponding author.*

- *This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences under grant XDA17040517, and by the Natural Science Foundation of China (NSFC) under grants 61871362 and U19B2023. The work of Miao Pan was supported in part by the U.S. National Science Foundation under grants US CNS-1801925 and CNS-2029569.*

Manuscript received X XX, XXXX; revised X XX, XXXX.

larger payload capacity. Therefore, the cooperation of HAPs and UAVs has the great potential to provide powerful services for IoT devices that work around the clock and are distributed across the ground, ocean, and air [12].

In this paper, we propose an aerial hierarchical MEC system composed of MEC-mounted HAPs and MEC-mounted UAVs in the air to provide computing services for the IoT devices covered by HAPs and UAVs. Each computing task is non-divisible owing to the dependency among the bits in the task [13]. Specifically, if a device cannot execute a computing task within its maximum tolerable delay, it can offload the task to a UAV for processing. If a UAV cannot execute an offloaded task while meeting the delay requirement, it can re-offload the task to an HAP for processing. In addition, we consider the queuing mechanism applied in the offloading and processing procedures of tasks. Assuming that each UAV has only one CPU due to its limited payload capacity, the processing capacity allocated to an offloaded task relies on the task load of the UAV. When numerous devices offload their tasks to the same UAV for processing, the high task load may cause these offloaded tasks to experience a long processing delay. Moreover, some offloaded tasks may even be abandoned if their deadlines expire. Hence, the processing delay of a task depends on the offloading decisions of the task itself and tasks that arrived previously.

This paper focuses on minimizing the long-term system cost which considers the processing delay of the tasks that are completely processed and the penalties for abandoned tasks. Traditional approaches such as branch-and-bound algorithms [14] and heuristic algorithms [15] mostly consider one-shot optimization and require prior knowledge of systems to construct corresponding task offloading strategies. These methods are difficult to apply in our proposed aerial hierarchical MEC system due to several stochastic factors, such as the number of tasks per device, the number of available UAVs, and the insertion of emergency tasks. To address these challenges, deep reinforcement learning (DRL) enables agents to learn the optimal policy according to local observations without prior knowledge of the system model. For a UAV, the tasks offloaded from IoT devices previously influence current task loads. Hence, the task load of each UAV follows a temporal correlation. In addition, an IoT device located in the overlapping region of adjacent UAVs can offload its task to one of them, which causes spatial correlation of task loads between adjacent UAVs. Therefore, convolutional LSTM (ConvLSTM) networks can be exploited to extract spatio-temporal features of task loads and predict the future task loads of all UAVs. Moreover, traditional DRL randomly samples experience transitions from replay memory, in which all the experiences are treated equally and their significance is ignored. However, ignoring the significance of useful experiences will cause the training process to suffer from poor stability and slow convergence.

To solve the above difficulties, this paper develops a multi-agent DRL-based computation offloading algorithm. By adopting the algorithm, each device can make its offloading decision according to local observations, including task properties, channel gains, predicted task loads, and queue information. In summary, the contributions of this paper are summarized as follows:

- We propose a novel aerial hierarchical MEC system composed of MEC-mounted HAPs and MEC-mounted UAVs. In the proposed system, we formulate a task offloading problem to minimize the long-term processing cost while satisfying the queueing mechanism in the offloading procedure and processing procedure of tasks.
- To achieve long-term cost minimization, we propose a multi-agent DRL-based computation offloading algorithm. The algorithm incorporates the ConvLSTM network and prioritized experience replay (PER) method. Specifically, the ConvLSTM network is utilized to estimate the future task loads of UAVs to improve the success ratio of completing tasks. The PER method guarantees that useful experiences are replayed with higher probabilities, which can not only increase the convergence performance but also improve the training stability.
- We perform simulations to analyze the performance of our proposed algorithm compared with those of three DQN variants, a random computation offloading method, a branch-and-bound-based algorithm, and a metaheuristic-based algorithm. The proposed algorithm outperforms other methods in most scenarios with different task arrival probabilities, different maximum tolerable delays, and different numbers of devices in terms of the average cost and ratio of dropped tasks.

The remainder of this paper is organized as follows. Section 2 introduces the related work. In section 3, we present the system model and formulate the optimization problem of computation offloading. The multi-agent DRL-based computation offloading algorithm is proposed in section 4, followed by the performance evaluation in section 5. Finally, conclusions are drawn in Section 6.

2 Related Work

Computation offloading in traditional three-tier MEC networks consisting of devices, edges, and cores has attracted much attention from academics and industries in recent years. The work in [16] investigated a dynamic service caching and task offloading strategy to minimize the computation latency under a long-term energy consumption constraint in MEC-enabled dense cellular networks. In [17], the authors utilized a randomized rounding technique to optimize the service placement and request routing with the goal of minimizing the load of the centralized cloud in dynamic dense MEC networks. Zhou *et al.* studied the joint optimization of computing offloading and service caching to minimize the task cost in edge computing-based smart grids [18]. The work in [19] jointly optimized the service caching, computation offloading, transmission, and computing resource allocation, aiming to minimize the overall computation and delay costs in a general scenario of multiple users with multiple tasks. Wang *et al.* formulated an optimization problem of task offloading and resource allocation as a Markov decision process with the goal of minimizing the delay and energy consumption costs in a three-tier MEC system [20]. In [21], an energy-efficient task offloading scheme was presented to minimize the wireless energy transmission in a three-tier wireless-powered MEC network. The traditional three-tier MEC network provides efficient service to users. However, the inherent defects of terrestrial infrastructures pose significant obstacles in realizing the immense potential benefits offered by IoT devices. In particular, it is expensive to deploy traditional three-tier MEC networks in remote areas with low population density, difficult terrain, and a lack of infrastructure, such as power grids. Moreover, terrestrial connectivity will be interrupted in some emergency scenarios, such as natural disasters.

Motivated by the above, the non-terrestrial edge computing network is a promising solution for providing various services to IoT devices working around the clock, which are distributed across the ground, ocean, and air. Aerial computing networks generally consist of two primary components: 1) low-altitude platforms (LAPs), such as UAVs and drones; and 2) HAPs, such as airships and HABs. There are abundant studies on LAP-based aerial computing systems. The work in [22] investigated a multi-dimensional resource management for UAV-assisted vehicle networks and maximized the number of tasks offloaded from vehicles by optimizing association decisions and resource allocations. In [23], Zhou *et al.* investigated the impact of task priority on MEC networks in which UAVs provide mobile computing services for users in multiple hotspots. The problem focuses on maximizing the utility of users by optimizing UAV hotspot selection and task offloading decisions. The authors of [24] optimized the trajectories, task allocation, and communication resource management of UAVs to minimize the sum of the execution delay and energy consumption. In [25], the authors proposed a spiral UAV placement algorithm for minimizing the number of UAVs to cover all users. Then, they utilized a DRL-based secure transmission method to maximize the system utility. A dynamic UAV edge computing network was developed in [26], which supports the dynamic entry and exit of UAVs and updates UAV deployment according to the user distribution. In [27], Liu *et al.* proposed a two-layer UAV maritime communication network and minimized the sum of the communication delay and computation delay by optimizing the trajectory of the centralized top-UAV and the configuration of virtual machines. A multi-agent path planning scheme was proposed in [28] to minimize the energy consumption of UAVs while considering the UAV load balance.

Several recent related works on HAP-based aerial computing have been presented. An HAP-enabled edge computing paradigm was presented in [29] to provide low-latency and high-efficiency connectivity for massive IoT users. The work in [30] focused on minimizing the energy and time consumption for task computation and transmission in an MEC-enabled balloon network where HABs act as wireless base stations. The work in [31] aimed at minimizing costs in the HAP-MEC-cloud network while ensuring communication, computing, and caching resource constraints. A column generation computation offloading algorithm and a greedy computation offloading algorithm are presented to solve the problem. In [32], the authors proposed an HAP-based computation framework for intelligent transportation systems (ITS) where the HAP stores the fundamental data for ITS-based applications to reduce the system delay.

There are also some works on multitier non-terrestrial

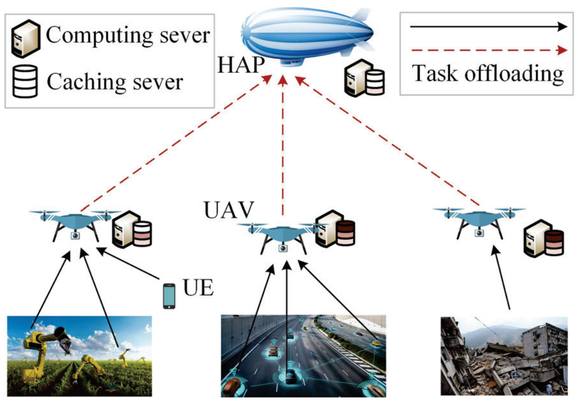


Fig. 1. Aerial hierarchical MEC system.

computing architecture. Ding *et al.* presented a satellite-HAP-integrated edge computing network to provide edge computing services for ground users and minimize the weighted sum of the energy consumption of the network [33]. In [34], an HAP and UAV cooperative framework was proposed to provide MEC services for IoT devices in disasters or remote areas, and a matching game theory based algorithm was presented to make the offloading decisions. Mao *et al.* focused on task computations in a space-aerial-assisted mixed cloud-edge computing framework composed of UAVs and satellites [35], where UAVs provide low-delay MEC services for IoT devices and satellites to guarantee ubiquitous cloud computing. The work in [36] presented a space-air-ground-integrated network and proposed a learning-based queue-aware algorithm to optimize task offloading and computation resource allocation. However, these works do not leverage the advantages of multiple and distributed network datasets, such as diverse channel features and environmental properties, while enhancing network quality. Moreover, the datasets of future intelligent aerial networks are distributed across large-scale networks instead of being centrally located. Hence, it is highly necessary to utilize distributed and large-scale optimization approaches for providing scalable and intelligent aerial computing applications. Compared with traditional terrestrial computing infrastructures, computing platforms in aerial computing networks possess limited storage resources and battery capabilities, which may hinder the deployment of aerial computing services. Hence, intelligent aerial computing is expected to become a dominant research area where artificial intelligence functionalities can be integrated into aerial devices at UAVs and HAPs to realize self-controlled and autonomous aerial systems. In this paper, we propose a distributed multi-agent DRL-based computation offloading method to optimize aerial hierarchical edge computing networks.

3 System Model

In this section, we first introduce the aerial hierarchical MEC system. Then the user device model, MEC-mounted UAV model, and MEC-mounted HAP model are presented. Finally, we formulate a task offloading problem with the goal of minimizing the total processing cost of tasks.

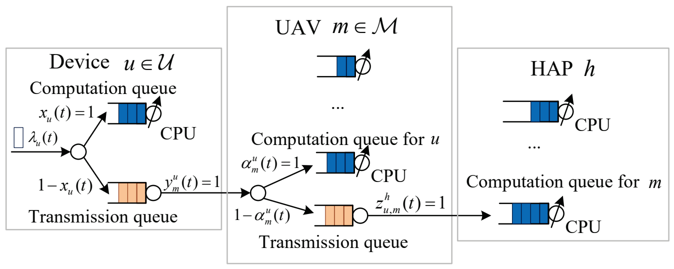


Fig. 2. The queue scheduling mechanism in the aerial hierarchical MEC system.

3.1 Aerial Hierarchical MEC System

As shown in Figure 1, our proposed aerial hierarchical MEC system consists of an HAP, multiple UAVs in the air, and abundant terrestrial user devices. Note that both the HAP and the UAVs remain quasi-stationary in this system. The set of UAVs is denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$, and all the UAVs fly within the coverage of the HAP. We assume that the HAP and UAVs are equipped with edge servers and that the loading capacity of the HAP is stronger than that of each UAV.

Some user devices can only execute lightweight computing tasks locally due to their limited computing capability. Hence, computation-intensive tasks can be offloaded to a surrounding UAV. If the task load of the UAV is extremely high, the offloaded tasks may be abandoned when their deadlines expire. In this case, the HAP can assist the overloaded UAV in executing the computing task. That is, the UAV regarded as a communication relay transmits the data offloaded from user devices to the HAP. Therefore, a computing task has three choices: 1) computed locally; 2) offloaded to a UAV and computed by the UAV; or 3) offloaded to the HAP through a UAV relay and computed by the HAP. Figure 2 illustrates the offloading procedure and processing procedure of tasks in the aerial hierarchical MEC system.

3.2 User Device Model

This paper focuses on a single episode with T time slots, i.e., $\mathcal{T} = \{1, 2, \ldots, T\}$, and the duration of each time slot is δ seconds. The set of user devices is denoted by $\mathcal{U} = \{1, 2, \ldots, U\}$. For each user device, a new task arrives with a Poisson distribution at the beginning of each time slot [37]. The computing task generated by user device $u \in \mathcal{U}$ at time slot $t \in \mathcal{T}$ is described as $\lambda_u(t) = \{\rho_u(t), \vartheta_u(t), \beta_u\}$, where $\lambda_u(t)$, $\rho_u(t)$, $\vartheta_u(t)$, and β_u denote the unique index of the task, the data size (in bits), the maximum tolerable delay (in time slots), and the required processing density (in CPU cycles/bit), respectively. The task $\lambda_u(t)$ is dropped if it has not been completely executed before the beginning of time slot $t + \vartheta_u(t)$.

Once task $\lambda_u(t)$ arrives at user device u at the beginning of time slot t, device u should make an offloading decision. The binary variable $x_u(t)$ indicates whether task $\lambda_u(t)$ is executed locally or not. If device u decides to execute task $\lambda_u(t)$ locally, $x_u(t) = 1$; otherwise, $x_u(t) = 0$. The value of binary variable $y_m^u(t)$ indicates whether task $\lambda_u(t)$ is offloaded to UAV m or not. If device u decides to offload task $\lambda_u(t)$ to UAV m, $y_m^u(t) = 1$; otherwise, $y_m^u(t) = 0$.

TABLE 1
Notation List

Notation	Description
$\mathcal{U}$	Set of user devices
$\mathcal{M}$	Set of UAVs
$\mathcal{T}$	Set of time slots
$\lambda_u(t)$	Computing task generated by device u
$\rho_u(t)$	Data size of task $\lambda_u(t)$
$\vartheta_u(t)$	Maximum tolerable delay of task $\lambda_u(t)$
β_u	Required processing density of task $\lambda_u(t)$
f_u	Computing capacity of device u
$\tau_u^{\text{com}}(t)$	Waiting time for processing task $\lambda_u(t)$
$q_u^{\text{tra}}(t)$	Length of transmission queue in device u
f_m^{UAV}	Computing capacity of UAV m
$q_{u,m}^{\text{UAV}}(t)$	Length of computation queue in UAV m
$\lambda_m^{\text{relay}}(t)$	Task in the transmission queue of UAV m
$\tau_m^{\text{relay}}(t)$	Waiting time for transmitting task $\lambda_m^{\text{relay}}(t)$
$q_m^{\text{relay}}(t)$	Length of transmission queue in UAV m
$\lambda_{m,h}^{\text{HAP}}(t)$	Task in the computation queue of the HAP
f_h^{HAP}	Computing capacity of the HAP
$\tau_{m,h}^{\text{HAP}}(t)$	Waiting time for processing task $\lambda_{m,h}^{\text{HAP}}(t)$
$q_{m,h}^{\text{HAP}}(t)$	Length of computation queue in the HAP

In addition, the binary variable $\alpha_m^u(t)$ denotes whether the offloaded task $\lambda_u(t)$ is computed by UAV m or not. If task $\lambda_u(t)$ is computed by UAV m, $\alpha_m^u(t) = 1$; otherwise, $\alpha_m^u(t) = 0$. We let binary variable $z_{u,m}^h(t)$ indicate whether task $\lambda_u(t)$ is offloaded to the HAP through the relay UAV m or not. If UAV m transmits the data of the offloaded task $\lambda_u(t)$ to the HAP, $z_{u,m}^h(t) = 1$; otherwise, $z_{u,m}^h(t) = 0$. Note that each computing task can be processed by one node, i.e.,

$$\sum_{m \in \mathcal{M}} y_m^u(t) = \mathbb{1}(x_u(t) = 0), \quad \forall u \in \mathcal{U}, \forall t \in \mathcal{T}, \tag{1}$$

$$\alpha_m^u(t) + z_{u,m}^h(t) = y_m^u(t), \quad \forall u \in \mathcal{U}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \tag{2}$$

where the indicator function $\mathbb{1}(x \in \mathcal{X}) = 1$ if $x \in \mathcal{X}$, and $\mathbb{1}(x \in \mathcal{X}) = 0$ otherwise. The notations used in this paper are listed in Table 1.

3.2.1 Local Computation Queue

The local computation queue of the user device follows a first-in-first-out (FIFO) pattern. Assuming that there is one CPU for processing tasks in the user device, a new arrival task must wait in the computation queue if previous tasks occupy the computing resource. For the task $\lambda_u(t)$ which has been placed in the local computation queue (i.e., $x_u(t) = 1$), $\tau_u^{\text{com}}(t)$ denotes the number of time slots spent in waiting for processing, and $d_u^{\text{com}}(t)$ represents the time slot when the task $\lambda_u(t)$ has either been completely executed or abandoned. Given $d_u^{\text{com}}(\acute{t})$ of previously arrived tasks $\lambda_u(\acute{t})$ for $\acute{t} < t$, user device u will compute $\tau_u^{\text{com}}(t)$ as follows:

$$\tau_u^{\text{com}}(t) = \max\left\{ \max_{\acute{t} \in \{0,1,\ldots,t-1\}} d_u^{\text{com}}(\acute{t}) - t, 0 \right\}. \tag{3}$$

Initially, $d_u^{\text{com}}(0)$ is set to zero. The time slot $d_u^{\text{com}}(t)$ can be computed as

$$d_u^{\text{com}}(t) = \min\left\{t + \tau_u^{\text{com}}(t) + \left\lceil \frac{\rho_u(t)\cdot\beta_u}{f_u\cdot\delta}\right\rceil, t + \vartheta_u(t)\right\}, \quad (4)$$

where $\lceil\cdot\rceil$ denotes the ceiling function and f_u is the computing capacity of the CPU (in CPU cycles/s) in device u. The term $t+\tau_u^{\text{com}}(t)$ represents the time slot when task $\lambda_u(t)$ starts being processed. The term $\lceil(\rho_u(t)\cdot\beta_u)/(f_u\cdot\delta)\rceil$ is the number of time slots spent in processing task $\lambda_u(t)$. Therefore, the first term in the minimization function represents the time slot when task $\lambda_u(t)$ will be completely executed if the deadline of the task is not considered. The second term denotes the time slot when task $\lambda_u(t)$ will be abandoned.

3.2.2 Local Transmission Queue

The transmission queue of each user device follows an FIFO pattern. The task placed in the transmission queue will be offloaded to a chosen UAV. Assuming that each user device transmits data on an orthogonal channel, the wireless communication between a device and a UAV suffers from line-of-light (LoS) path loss. We define $g_{u,m}(t)$ to denote the channel gain between device u and UAV m during time slot t. The transmission data rate from device u to UAV m (in bits/s) is obtained as follows:

$$r_{u,m}^{\text{tra}}(t) = B_u \log_2\left(1 + \frac{g_{u,m}(t)\cdot P_u}{N_0}\right), \quad (5)$$

where B_u is the communication bandwidth allocated to a single channel. The notation P_u denotes the transmitting power of device u and N_0 is the noise power at UAV m. We assume that $r_{u,m}^{\text{tra}}(t)$ is constant during time slot t.

If task $\lambda_u(t)$ is placed in the local transmission queue at the beginning of time slot t (i.e., $x_u(t)=0$), we let $d_u^{\text{tra}}(t)$ represent the time slot when task $\lambda_u(t)$ has been completely transmitted. Due to the uncertain future data rate between device u and UAV m, the value of $d_u^{\text{tra}}(t)$ cannot be obtained until task $\lambda_u(t)$ is completely transmitted. We define $\bar{d}_u^{\text{tra}}(t)$ to denote the time slot when the transmission of task $\lambda_u(t)$ starts, i.e.,

$$\bar{d}_u^{\text{tra}}(t) = \max\left\{\max_{\acute{t}\in\{0,1,\ldots,t-1\}} d_u^{\text{tra}}(\acute{t}),\ t\right\}. \quad (6)$$

Initially, $\bar{d}_u^{\text{tra}}(0)$ is set to zero. In addition, the values of $\bar{d}_u^{\text{tra}}(t)$ and $d_u^{\text{tra}}(t)$ must satisfy the following constraints:

$$\sum_{\acute{t}=\bar{d}_u^{\text{tra}}(t)}^{d_u^{\text{tra}}(t)-1} \sum_{m\in\mathcal{M}} y_m^u(t)\cdot r_{u,m}^{\text{tra}}(\acute{t})\cdot\delta < \rho_u(t), \quad (7)$$

$$\sum_{\acute{t}=\bar{d}_u^{\text{tra}}(t)}^{d_u^{\text{tra}}(t)} \sum_{m\in\mathcal{M}} y_m^u(t)\cdot r_{u,m}^{\text{tra}}(\acute{t})\cdot\delta \geq \rho_u(t). \quad (8)$$

Specifically, such inequations mean that the data size of task $\lambda_u(t)$ is larger than the size of the data transmitted from $\bar{d}_u^{\text{tra}}(t)$ to $d_u^{\text{tra}}(t)-1$ and is no larger than that from time slot $\bar{d}_u^{\text{tra}}(t)$ to $d_u^{\text{tra}}(t)$.

Let $q_u^{\text{tra}}(t)$ (in bits) denote the length of the transmission queue in device u at the end of time slot t. If a task in the transmission queue of device u is being transmitted to a UAV during time slot t, we denote the task as $\tilde{\lambda}_u(t)$. The amount of data transmitted by device u during time slot t is defined as

$$\varphi_u(t) = \sum_{\acute{t}=1}^{t}\sum_{m\in\mathcal{M}} \mathbb{1}\left(\tilde{\lambda}_u(t) = \lambda_u(\acute{t})\right)\cdot y_m^u(\acute{t})\cdot r_{u,m}^{\text{tra}}(t)\cdot\delta. \quad (9)$$

Therefore, the value of $q_u^{\text{tra}}(t)$ can be updated as

$$q_u^{\text{tra}}(t) = \max\Big\{q_u^{\text{tra}}(t-1) + \rho_u(t)\cdot\mathbb{1}\left(x_u(t)=0\right) - \xi_u(t) - \varphi_u(t),\ 0\Big\}, \quad (10)$$

where $\xi_u(t)$ (in bits) represents the amount of data dropped by the transmission queue at the end of time slot t.

3.3 MEC-mounted UAV Model

We let $\mathcal{U}_m(t)\subset\mathcal{U}$ denote the set of user devices under the coverage of UAV m during time slot t. Each MEC-mounted UAV $m\in\mathcal{M}$ maintains $|\mathcal{U}_m(t)|$ computation queues and a transmission queue. Each computation queue corresponds to a user device in the set $\mathcal{U}_m(t)$. The transmission queue in each UAV is responsible for sending the data of tasks offloaded from user devices to the HAP.

3.3.1 Computation Queues of UAV

Each computation queue of a UAV follows an FIFO pattern. If UAV m receives an offloaded task from device $u\in\mathcal{U}_m(t-1)$ during time slot $t-1$ and is required to process the task, we denote the task as $\lambda_{u,m}^{\text{UAV}}(t)$ and place it in the corresponding computation queue at the beginning of time slot t. Specifically, if task $\lambda_u(\acute{t})$ for $\acute{t}\in\{1,2,\ldots,t-1\}$ is received at UAV m during time slot $t-1$ (i.e., $y_m^u(\acute{t})=1$) and is processed by UAV m (i.e., $\alpha_m^u(\acute{t})=1$), then $\lambda_u(\acute{t})=\lambda_{u,m}^{\text{UAV}}(t)$. In addition, the number of bits put into the computation queue of UAV m corresponding to device u at the beginning of time slot t is denoted by $\rho_{u,m}^{\text{UAV}}(t)$.

For each UAV m, let $q_{u,m}^{\text{UAV}}(t)$ (in bits) represent the length of the computation queue corresponding to device u at the end of time slot t. Among those computation queues, we define a nonempty queue during time slot t if either there is a new arrival task reaching the queue at the beginning of time slot t (i.e., $\rho_{u,m}^{\text{UAV}}(t)>0$) or if the task in the queue was not processed to completion at the end of time slot $t-1$ (i.e., $q_{u,m}^{\text{UAV}}(t-1)>0$). Let $\mathbf{\Psi}_m(t)$ indicate the set of nonempty computation queues at UAV m during time slot t. Then, we have

$$\mathbf{\Psi}_m(t) = \{u | u\in\mathcal{U}_m(t-1), \rho_{u,m}^{\text{UAV}}(t)>0 \text{ or } q_{u,m}^{\text{UAV}}(t-1)>0\}. \quad (11)$$

Let $\Psi_m(t)$ denote the number of nonempty computation queues at UAV m during time slot t, i.e., $\Psi_m(t)=|\mathbf{\Psi}_m(t)|$.

Assuming that each MEC-mounted UAV has one CPU, the computing capacity of UAV m is allocated to the nonempty computation queues at UAV m equally. We denote the computing capacity of the CPU (in CPU cycles/s) at UAV m as f_m^{UAV}. The queue length $q_{u,m}^{\text{UAV}}(t)$ can be computed as

$$q_{u,m}^{\text{UAV}}(t) = \max\Big\{q_{u,m}^{\text{UAV}}(t-1) + \rho_{u,m}^{\text{UAV}}(t) - \xi_{u,m}^{\text{UAV}}(t)$$

$$- \frac{f_m^{\text{UAV}} \cdot \delta}{\beta_u \cdot \Psi_m(t)} \mathbb{1}\left(u \in \mathbf{\Psi}_m(t)\right), 0 \Bigg\}, \quad (12)$$

where $\xi_{u,m}^{\text{UAV}}(t)$ (in bits) is the amount of data abandoned by the queue at the end of time slot t.

If UAV m receives an offloaded task $\lambda_{u,m}^{\text{UAV}}(t)$ from device u and places the task in the corresponding computation queue at the beginning of time slot t, a variable $d_{u,m}^{\text{com}}(t) \in \mathcal{T}$ is defined to describe the time slot when task $\lambda_{u,m}^{\text{UAV}}(t)$ has been completely executed by UAV m. Since the value of $\Psi_m(t)$ varies with the offloading decisions of the tasks, the computing capacity allocated to each computation queue is unknown a priori for user devices. Hence, the value of $d_{u,m}^{\text{com}}(t)$ cannot be obtained until task $\lambda_{u,m}^{\text{UAV}}(t)$ is completely executed. We define $\bar{d}_{u,m}^{\text{com}}(t)$ to indicate the time slot when the processing of task $\lambda_{u,m}^{\text{UAV}}(t)$ starts, i.e.,

$$\bar{d}_{u,m}^{\text{com}}(t) = \max\left\{ \max_{t' \in \{0,1,\ldots,t-1\}} d_{u,m}^{\text{com}}(t'),\ t \right\}. \quad (13)$$

Initially, $\bar{d}_{u,m}^{\text{com}}(0)$ is set to zero. In addition, the values of $\bar{d}_{u,m}^{\text{com}}(t)$ and $d_{u,m}^{\text{com}}(t)$ must satisfy the following constraints:

$$\sum_{t'=\bar{d}_{u,m}^{\text{com}}(t)}^{d_{u,m}^{\text{com}}(t)-1} \frac{f_m^{\text{UAV}} \cdot \delta}{\beta_u \cdot \Psi_m(t')} \mathbb{1}\left(u \in \mathbf{\Psi}_m(t')\right) < \rho_{u,m}^{\text{UAV}}(t), \quad (14)$$

$$\sum_{t'=\bar{d}_{u,m}^{\text{com}}(t)}^{d_{u,m}^{\text{com}}(t)} \frac{f_m^{\text{UAV}} \cdot \delta}{\beta_u \cdot \Psi_m(t')} \mathbb{1}\left(u \in \mathbf{\Psi}_m(t')\right) \geq \rho_{u,m}^{\text{UAV}}(t). \quad (15)$$

Specifically, such inequations mean that the data size of task $\lambda_{u,m}^{\text{UAV}}(t)$ is larger than the data size processed by UAV m from time slot $\bar{d}_{u,m}^{\text{com}}(t)$ to $d_{u,m}^{\text{com}}(t) - 1$ and is no larger than that from time slot $\bar{d}_{u,m}^{\text{com}}(t)$ to $d_{u,m}^{\text{com}}(t)$.

3.3.2 *Transmission Queue of UAV*

The transmission queue of a UAV follows an FIFO pattern. In addition, if the UAV receives more than one task offloaded from different user devices during the same time slot and these tasks are then offloaded to the HAP, the task with the shortest remaining time has the highest priority to be put into the transmission queue. Assuming that each UAV transmits data to the HAP on an orthogonal channel, we define $g_{m,h}$ to denote the channel gain between UAV m and the HAP. The transmission data rate from UAV m to the HAP (in bits/s) can be formulated as

$$r_{m,h}^{\text{tra}} = B_m \log_2\left(1 + \frac{g_{m,h} P_m}{k_B T_s B_m}\right), \quad (16)$$

where B_m denotes the communication bandwidth allocated to a single channel and P_m is the transmitting power of UAV m. Let k_B and T_s denote Boltzmann's constant and the system noise temperature, respectively. We assume that the value of $r_{m,h}^{\text{tra}}$ is constant.

If UAV m receives an offloaded task from device u during time slot $t-1$ and places the task in the transmission queue at the beginning of time slot t, we denote the task as $\lambda_m^{\text{relay}}(t)$. Specifically, if task $\lambda_u(t')$ for $t' \in \{1, 2, \ldots, t-1\}$ arrives at UAV m during time slot $t-1$ (i.e., $y_m^u(t') = 1$ and $\alpha_m^u(t') = 0$) and has the highest priority, then $\lambda_m^{\text{relay}}(t) = \lambda_u(t')$. The maximum tolerable delay and the data size of task $\lambda_m^{\text{relay}}(t)$ are denoted by $\vartheta_m^{\text{relay}}(t) = \vartheta_u(t') - t + t'$ and $\rho_m^{\text{relay}}(t) = \rho_u(t')$, respectively. We denote $\tau_m^{\text{relay}}(t)$ as the number of time slots spent in waiting for transmission, and let $d_m^{\text{relay}}(t)$ represent the time slot when task $\lambda_m^{\text{relay}}(t)$ has either been transmitted completely or abandoned. Given $d_m^{\text{relay}}(t')$ for $t' < t$, UAV m computes $\tau_m^{\text{relay}}(t)$ as follows:

$$\tau_m^{\text{relay}}(t) = \max\left\{ \max_{t' \in \{0,1,\ldots,t-1\}} d_m^{\text{relay}}(t') - t, 0 \right\}. \quad (17)$$

Initially, $d_m^{\text{relay}}(0)$ is set to zero. Then, the value of $d_m^{\text{relay}}(t)$ can be expressed as

$$d_m^{\text{relay}}(t) = \min\left\{ t + \tau_m^{\text{relay}}(t) + \left\lceil \frac{\rho_m^{\text{relay}}(t)}{r_{m,h}^{\text{tra}} \cdot \delta} \right\rceil, t + \vartheta_m^{\text{relay}}(t) \right\}. \quad (18)$$

The term $t + \tau_m^{\text{relay}}(t)$ represents the time slot when task $\lambda_m^{\text{relay}}(t)$ starts being transmitted to the HAP. The term $\lceil \rho_m^{\text{relay}}(t)/(r_{m,h}^{\text{tra}} \cdot \delta) \rceil$ is the number of time slots required to transmit the data of task $\lambda_m^{\text{relay}}(t)$. Let $q_m^{\text{relay}}(t)$ (in bits) denote the length of the transmission queue in UAV m at the end of time slot t. The value of $q_m^{\text{relay}}(t)$ can be updated as

$$q_m^{\text{relay}}(t) = \max\left\{ q_m^{\text{relay}}(t-1) + \rho_m^{\text{relay}}(t) - r_{m,h}^{\text{tra}} \cdot \delta - \xi_m^{\text{relay}}(t), 0 \right\}, \quad (19)$$

where $\xi_m^{\text{relay}}(t)$ (in bits) represents the amount of data abandoned by the transmission queue of UAV m at the end of time slot t.

3.4 MEC-mounted HAP Model

The MEC-mounted HAP maintains M computation queues, and each queue corresponds to a UAV in set $\mathcal{M}$. In addition, each computation queue of the HAP follows an FIFO pattern. If the HAP receives an offloaded task from UAV $m \in \mathcal{M}$ during time slot $t-1$ and places the task in the corresponding queue at the beginning of time slot t, we denote the task as $\lambda_{m,h}^{\text{HAP}}(t)$. Specifically, if task $\lambda_m^{\text{relay}}(t')$ for $t' \in \{1, 2, \ldots, t-1\}$ offloaded from UAV m arrives at the HAP during time slot $t-1$, then $\lambda_m^{\text{relay}}(t') = \lambda_{m,h}^{\text{HAP}}(t)$. The number of bits put into the computation queue of HAP h corresponding to UAV m at the beginning of time slot t is denoted by $\rho_{m,h}^{\text{HAP}}(t)$. We let $\beta_{m,h}^{\text{HAP}}(t)$ represent the processing density of task $\lambda_{m,h}^{\text{HAP}}(t)$. The maximum tolerance delay of task $\lambda_{m,h}^{\text{HAP}}(t)$ is $\vartheta_{m,h}^{\text{HAP}}(t) = \vartheta_m^{\text{relay}}(t') - t + t'$.

Assuming that the MEC-mounted HAP has M CPUs, each CPU is responsible for processing the tasks in a computation queue. The computing capacity of a CPU (in CPU cycles/s) in the HAP is denoted as f_h^{HAP}. We define a variable $d_{m,h}^{\text{HAP}}(t)$ to represent the time slot when task $\lambda_{m,h}^{\text{HAP}}(t)$ has been either completely executed or abandoned by HAP h. In addition, the variable $\tau_{m,h}^{\text{HAP}}(t)$ denotes the number of time slots spent in waiting for the processing task $\lambda_{m,h}^{\text{HAP}}(t)$. Given $d_{m,h}^{\text{HAP}}(t')$ for $t' < t$, the value of $\tau_{m,h}^{\text{HAP}}(t)$ can be computed by

$$\tau_{m,h}^{\text{HAP}}(t) = \max\left\{ \max_{t' \in \{0,1,\ldots,t-1\}} d_{m,h}^{\text{HAP}}(t') - t, 0 \right\}. \quad (20)$$

Initially, $d_{m,h}^{\text{HAP}}(0)$ is set to zero. The value of $d_{m,h}^{\text{HAP}}(t)$ is calculated by

$$d_{m,h}^{\text{HAP}}(t) = \min\left\{t + \tau_{m,h}^{\text{HAP}}(t) + \left\lceil \frac{\rho_{m,h}^{\text{HAP}}(t) \cdot \beta_{m,h}^{\text{HAP}}(t)}{f_h^{\text{HAP}} \cdot \delta} \right\rceil, \right.$$
$$\left. t + \vartheta_{m,h}^{\text{HAP}}(t) \right\}. \tag{21}$$

Let $q_{m,h}^{\text{HAP}}(t)$ (in bits) denote the length of the computation queue in the HAP corresponding to UAV m at the end of time slot t. The value of $q_{m,h}^{\text{HAP}}(t)$ can be updated as

$$q_{m,h}^{\text{HAP}}(t) = \max\left\{ q_{m,h}^{\text{HAP}}(t-1) + \rho_{m,h}^{\text{HAP}}(t) - \xi_{m,h}^{\text{HAP}}(t) \right.$$
$$\left. - \frac{f_h^{\text{HAP}} \cdot \delta}{\beta_{m,h}^{\text{HAP}}(t)}, 0 \right\}, \tag{22}$$

where $\xi_{m,h}^{\text{HAP}}(t)$ (in bits) denotes the amount of data abandoned by the queue at the end of time slot t.

3.5 Problem Formulation

If a computing task has been executed, the cost of the task is defined as the duration between the time slot when the task is generated and the time slot when the task has been completely executed. Let $c_u(t)$ (in time slots) represent the processing cost of task $\lambda_u(t)$. If task $\lambda_u(t)$ is executed at device u locally, we have

$$c_u(t) = d_u^{\text{com}}(t) - t. \tag{23}$$

If task $\lambda_u(t)$ is offloaded to some UAV for processing, then

$$c_u(t) = \sum_{m\in\mathcal{M}} \sum_{\acute{t}=t}^{T} \mathbb{1}(\lambda_u(t) = \lambda_{u,m}^{\text{UAV}}(\acute{t})) \cdot d_{u,m}^{\text{com}}(\acute{t}) - t. \tag{24}$$

Otherwise, if task $\lambda_u(t)$ is offloaded to HAP h for processing via a UAV, we have

$$c_u(t) = \sum_{m\in\mathcal{M}} \sum_{\acute{t}=t}^{T} \sum_{t^*=\acute{t}}^{T} \mathbb{1}(\lambda_u(t) = \lambda_m^{\text{relay}}(\acute{t})) \cdot$$
$$\mathbb{1}(\lambda_m^{\text{relay}}(\acute{t}) = \lambda_{m,h}^{\text{HAP}}(t^*)) \cdot d_{m,h}^{\text{HAP}}(t^*) - t. \tag{25}$$

However, computing task $\lambda_u(t)$ may be abandoned if it is not fully processed when its deadline expires. In this case, the processing cost of task $\lambda_u(t)$ is defined as $c_u(t) = C$, where $C > 0$ is a constant penalty. According to this discussion, the total cost in the aerial hierarchical MEC system is formulated as

$$c_{\text{tot}} = \sum_{t\in\mathcal{T}} \sum_{u\in\mathcal{U}} c_u(t). \tag{26}$$

The objective of this paper is to find an optimal computation offloading decision to minimize the total system cost c_{tot} while satisfying the queueing mechanism in the offloading procedure and processing procedure of tasks. We formulate the optimization problem as

$$\begin{aligned}
\mathbf{P}: \quad & \min\ c_{\text{tot}} \\
\text{s.t.}\ & x_u(t) \in \{0,1\},\ \forall u \in \mathcal{U}, \forall t \in \mathcal{T}, \\
& y_m^u(t) \in \{0,1\},\ \forall u \in \mathcal{U}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \\
& \alpha_m^u(t) \in \{0,1\},\ \forall u \in \mathcal{U}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \\
& z_{u,m}^h(t) \in \{0,1\},\ \forall u \in \mathcal{U}, \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \\
& (1), (2), (7), (8), (14), (15), (23), (24), (25).
\end{aligned}$$

Theorem 1. *The three-tier computation offloading problem* **P** *is NP-hard.*

Proof. We prove the NP-hardness of the three-tier computation offloading problem **P** by introducing an instance of multiple knapsack problem with assignment restriction (MKAR), which is NP-hard. According to the definition in [38], given a set of items with associated positive real profits and a set of knapsacks in which each has a positive real capacity, the MKAR allocates items to knapsacks with the aim of maximizing total profit while satisfying knapsack capacity limits and assignment restrictions.

Then, we reduce the known NP-hard MKAR to the three-tier computation offloading problem **P**. Each task $\lambda_u(t)$ (item in MKAR) has a data size $\rho_u(t)$ and a negative computation cost $-c_u(t)$ (profit in MKAR). Then, the problem **P** allocates each task to an appropriate MEC node (local device, UAV, or HAP) to maximize the total profit $\sum_{t\in\mathcal{T}}\sum_{u\in U} -c_u(t)$ while satisfying the UAV storing capacity limit (knapsack capacity limit in MKAR) and task offloading constraints in the aerial hierarchical MEC system (assignment restriction in MKAR).

Therefore, each instance of the NP-hard MKAR is polynomial-time reducible to the three-tier computation offloading problem **P**. Hence, the problem **P** is NP-hard. □

Designing a distributed computation offloading algorithm poses challenges in our proposed aerial hierarchical edge computing network. Since the optimization problem **P** is demonstrated to be NP-hard, solving this problem via conventional optimization approaches is difficult because of the complicated interactions among the tasks and the dynamic computing system. On the other hand, when an IoT device makes an offloading decision, it does not know a priori knowledge of task loads at MEC-mounted UAVs. This is because the load level relies on the offloading decisions of other IoT devices. Moreover, the load levels at UAVs may vary over time. Thus, the queuing delay and processing delay of a new arrival task will be affected by the decisions of predecessor tasks generated at other devices. Nonetheless, deep reinforcement learning (DRL) based algorithms are promising for solving the dynamic computation offloading problem. Thus, we propose utilizing the DRL technique, which can enable IoT devices to make offloading decisions according to local observations without the prior knowledge of the system model or dynamic changes.

4 DRL-based Computation Offloading

In this section, we first propose a ConvLSTM network to predict the future task loads of UAVs. Then, a multi-agent DRL-based computation offloading algorithm featuring PER is presented to enable each device to make its offloading decision.

4.1 ConvLSTM Network

As explained in Section 1, the task loads of UAVs have a spatio-temporal correlated relationship. On the one hand, IoT devices may offload tasks to nearby MEC-mounted UAVs. Hence, UAVs above an area with massive IoT devices will meet a great number of demands. Due to the movement of some IoT devices (e.g., connected vehicles and autonomous delivery fleets), the computation demands on MEC-mounted UAVs above different areas change accordingly. That is, the demands on a UAV regarded as the task load of the UAV will follow a spatial correlation. In addition, an IoT device located in the overlapping coverage region of adjacent UAVs can offload its task to one of them, which also causes the spatial correlation of task loads between adjacent UAVs. On the other hand, with the emergence of various compute-intensive applications (e.g., onboard AR/VR entertainment, autonomous driving, and autonomous aerial delivery), the processing and transmission of such a task may continue for multiple time slots. In addition, for a UAV, the tasks offloaded from IoT devices previously influence the current task load of the UAV. Therefore, the task load of each UAV follows a temporal correlation. More importantly, the task loads of all UAVs determine the completion ratio of offloaded tasks in the network. However, the knowledge of task loads is unknown to IoT devices, and the fully connected deep neural network in traditional DRL lacks the representative ability to make accurate state inferences.

To solve such problems, we propose utilizing the ConvLSTM network to extract spatio-temporal features of UAV task loads and predict task loads in the near future. Specifically, the ConvLSTM network structurally consists of convolutional neural networks (CNNs) and long short-term memory (LSTM) networks. CNNs have been widely applied in object detection and image recognition [39]. It can extract spatial features of an input two-dimensional tensor by several layers of convolutional operations. In each time slot, the task loads of all UAVs can be described as a map, where the value of each grid denotes the task load corresponding to the UAV in the grid. Then, the convolution operation is responsible for capturing the relationship between adjacent grids, i.e., the task loads of adjacent UAVs.

Moreover, the maps during continuous time slots form a time series. LSTM is capable of capturing temporal features and learning the long-term dependencies of sequential data [40]. There are three critical components in LSTM: the input gate, forget gate, and output gate. These gates cooperate to capture useful information from input data, which can avoid gradient vanish and explosion appearing in classical recurrent neural network. Hence, the relationships among the previous input vectors can be extracted. Based on the structure of the LSTM network, we replace the dot product in the gates with a convolution operation to construct the ConvLSTM framework.

Let two-dimensional tensor $\mathbf{\Psi}(t)$ represent the task loads of all UAVs at time slot t. The size of $\mathbf{\Psi}(t)$ is $l_r \times l_c$. We denote $\{\Psi(t)\}_{i,j}$ as the (i,j) element of $\mathbf{\Psi}(t)$, which equals the task load of UAV m located at the corresponding grid on the map during time slot t. To obtain the history of the task loads, we suppose that each UAV broadcasts its task load to the HAP at the end of each time slot. The

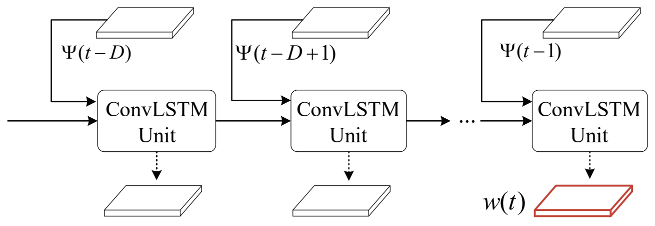


Fig. 3. The framework of the ConvLSTM network.

HAP utilizes the sequential data of task loads to predict the future task load of each UAV in the next time slot. Figure 3 illustrates the structure of a ConvLSTM network. The ConvLSTM network includes D ConvLSTM units, each of which takes the two-dimensional tensor data at one time slot as the input. These ConvLSTM units are connected in sequence to capture the variations in the sequential input tensor from $\mathbf{\Psi}(t-D)$ to $\mathbf{\Psi}(t-1)$. The final ConvLSTM unit outputs a two-dimensional tensor $\boldsymbol{w}(t)$, which denotes the predicted task loads of all the UAVs on the map during time slot t.

4.2 Three Basic Elements

Assuming that each user device is regarded as an agent, we introduce the observation/state, action, and reward for each agent as follows.

4.2.1 *State*

We assume that the HAP and each UAV broadcast the length of each computation queue and the length of each transmission queue at the end of each time slot. The state of user device $u \in \mathcal{U}$ in time slot t, denoted by $\boldsymbol{o}_u(t)$, includes task properties and environment-related information of the aerial hierarchical MEC system. Specifically, device u observes the following state at time slot t:

$$\boldsymbol{o}_u(t) = (\lambda_u(t), \boldsymbol{g}(t), d_u^{\text{com}}(t), q_u^{\text{tra}}(t), \widetilde{\boldsymbol{w}}(t), \boldsymbol{q}_u^{\text{UAV}}(t-1), \boldsymbol{q}^{\text{relay}}(t-1), \boldsymbol{q}^{\text{HAP}}(t-1)),$$

where $\boldsymbol{g}(t) = \{g_{u,m}(t), g_{m,h} | \forall m \in \mathcal{M}\}$ denotes the wireless channel gains at time slot t. A device agent can obtain its own signal channel gain $g_{u,m}(t)$ for all $m \in \mathcal{M}$ and the constant channel gain between each UAV and the HAP. The two-dimensional data $\boldsymbol{w}(t)$ are reshaped into one-dimensional data $\widetilde{\boldsymbol{w}}(t)$ and transmitted to each user device from the HAP. The length of the computation queue corresponding to device u in UAVs at the end of time slot $t-1$ is denoted by $\boldsymbol{q}_u^{\text{UAV}}(t-1) = \{q_{u,m}^{\text{UAV}}(t-1) | \forall m \in \mathcal{M}\}$. The length of the transmission queue in UAVs at the end of time slot $t-1$ is denoted by $\boldsymbol{q}^{\text{relay}}(t-1) = \{q_m^{\text{relay}}(t-1) | \forall m \in \mathcal{M}\}$. The length of the computation queue in the HAP at the end of time slot $t-1$ is denoted by $\boldsymbol{q}^{\text{HAP}}(t-1) = \{q_{m,h}^{\text{HAP}}(t-1) | \forall m \in \mathcal{M}\}$.

4.2.2 *Action*

Based on the observation $\boldsymbol{o}_u(t)$, device u will take action for a new arrival task $\lambda_u(t)$: (a) whether to execute the task locally or offload it to a UAV, i.e., $x_u(t)$; (b) which UAV the task is offloaded to, i.e., $\boldsymbol{y}_u(t) = \{y_m^u(t) | \forall m \in \mathcal{M}\}$; (c)

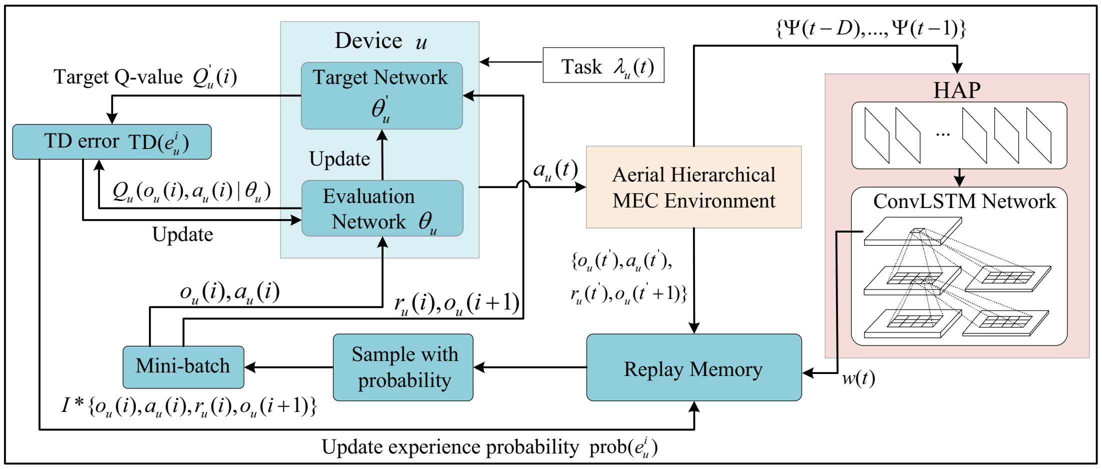

Device u
Target Network $\theta_u^{'}$
Update
Evaluation Network θ_u
Task $\lambda_u(t)$
Target Q-value $Q_u^{'}(i)$
TD error TD(e_u^i)
$Q_u(o_u(i), a_u(i) | \theta_u)$
Update
$a_u(t)$
Aerial Hierarchical MEC Environment
$\{\Psi(t-D),...,\Psi(t-1)\}$
HAP
ConvLSTM Network
$\{o_u(t^{'}), a_u(t^{'}), r_u(t^{'}), o_u(t^{'}+1)\}$
$o_u(i), a_u(i)$
$r_u(i), o_u(i+1)$
Mini-batch
Sample with probability
Replay Memory
$w(t)$
$I * \{o_u(i), a_u(i), r_u(i), o_u(i+1)\}$
Update experience probability prob(e_u^i)

Fig. 4. Overview of the multi-agent DRL-based computation offloading scheme. The right part with a red block is the ConvLSTM network deployed in the HAP. The left part with blue blocks is the DRL-based network deployed in each device.

whether to process the offloaded task in the selected UAV, i.e., $\boldsymbol{\alpha}_u(t) = \{\alpha_m^u(t) | \forall m \in \mathcal{M}\}$, or offload it to the HAP, i.e., $\boldsymbol{z}_u(t) = \{z_{u,m}^h(t) | \forall m \in \mathcal{M}\}$. Therefore, the action of device u in time slot t can be formulated as

$$\boldsymbol{a}_u(t) = \{x_u(t), \boldsymbol{y}_u(t), \boldsymbol{\alpha}_u(t), \boldsymbol{z}_u(t)\}. \tag{27}$$

We denote the action space as $\mathbb{A}_u$.

4.2.3 Reward

Considering the state $\boldsymbol{o}_u(t)$, device u takes the offloading policy $\boldsymbol{a}_u(t)$ to interact with the aerial hierarchical MEC environment and obtains the reward. The long-term reward of agent u is defined as

$$r_u(t) = \sum_{t=1}^{T} \gamma^t \cdot r(\boldsymbol{o}_u(t), \boldsymbol{a}_u(t)), \tag{28}$$

where the discount factor γ represents the importance of the reward. The value of $r(\boldsymbol{o}_u(t), \boldsymbol{a}_u(t))$ is set to be $\bar{C} - c_u(t)$, in which $\bar{C}$ denotes a constant larger than the upper bound of $c_u(t)$.

4.3 DRL-based Computation Offloading Algorithm

Our network consists of multiple IoT devices and multiple computing nodes (i.e., UAVs and HAPs). Utilizing all historical empirical data to train a single model centrally will cause IoT devices to lose many important local features and become incapable of capturing interaction features among devices. This may cause IoT agents to make erroneous computation offloading decisions according to their local observations and be unadaptable to the environments of our proposed aerial hierarchical edge computing network. To solve this problem, we propose utilizing multi-agent DRL, which is more suitable for complicated aerial hierarchical edge computing networks, including multiple IoT devices and multiple computing nodes.

In our proposed multi-agent DRL-based computation offloading scheme, each device $u \in \mathcal{U}$ is regarded as an agent. Assuming that device u maintains a replay memory R_u, the memory R_u stores the experience $(\boldsymbol{o}_u(t), \boldsymbol{a}_u(t), r_u(t), \boldsymbol{o}_u(t+1))$ of device u at the end of time slot t. In addition, each device u maintains two neural networks, namely a target network and an evaluation network. The parameters of the target network and the evaluation network are denoted by $\boldsymbol{\theta}_u^{'}$ and $\boldsymbol{\theta}_u$, respectively. Figure 4 illustrates the architecture of the DRL-based computation offloading scheme featured with the ConvLSTM network and PER method.

Assuming that the number of episodes is E, the initial state of device $u \in \mathcal{U}$ at the beginning of each episode is given by

$$\begin{aligned}\boldsymbol{o}_u(1) =&(\lambda_u(1), \boldsymbol{g}(1), d_u^{\text{com}}(1), q_u^{\text{tra}}(1), \widetilde{\boldsymbol{w}}(1),\\ &\boldsymbol{q}_u^{\text{UAV}}(0), \boldsymbol{q}^{\text{relay}}(0), \boldsymbol{q}^{\text{HAP}}(0)),\end{aligned} \tag{29}$$

where $\widetilde{\boldsymbol{w}}(1)$, $\boldsymbol{q}_u^{\text{UAV}}(0)$, $\boldsymbol{q}^{\text{relay}}(0)$, and $\boldsymbol{q}^{\text{HAP}}(0)$ are zero vectors. If device u has a new arrival task $\lambda_u(t)$ at the beginning of time slot t, it will take its action as follows

$$\boldsymbol{a}_u(t) = \begin{cases} \text{a random action from } \mathbb{A}_u, & \text{w.p. } \varepsilon, \\ \arg\max\limits_{\boldsymbol{a} \in \mathbb{A}_u} Q_u\left(\boldsymbol{o}_u(t), \boldsymbol{a} | \boldsymbol{\theta}_u\right), & \text{w.p. } 1-\varepsilon, \end{cases} \tag{30}$$

where ε denotes the probability of random exploration and $Q_u\left(\boldsymbol{o}_u(t), \boldsymbol{a} | \boldsymbol{\theta}_u\right)$ represents the Q-value of the evaluation network under the observed state $\boldsymbol{o}_u(t)$ and action $\boldsymbol{a}$.

Once device u performs action $\boldsymbol{a}_u(t)$, it observes the next state $\boldsymbol{o}_u(t+1)$ at the beginning of time slot $t+1$. Since multiple time slots may be needed to process and transmit a task, the reward $r_u(t)$ of task $\lambda_u(t)$ may not be obtained immediately. Therefore, at the beginning of time slot $t+1$, device u may obtain a set of rewards corresponding to some tasks that arrived before time slot t. Let $\mathcal{T}_{u,t}^* \subset \mathcal{T}$ denote the set of time slots such that each task $\lambda_u(\acute{t})$ was generated at the beginning of time slot $\acute{t} \in \mathcal{T}_{u,t}^*$ and then has been processed completely or abandoned in time slot t. Hence, $\mathcal{T}_{u,t}^*$ is formulated as

$$\mathcal{T}_{u,t}^* = \Bigg[\acute{t} \Bigg| \acute{t} = 1, 2, \ldots, t, \rho_u(\acute{t}) > 0,\ d_u^{\text{com}}(\acute{t}) = t$$
$$\text{or} \sum_{m \in \mathcal{M}} \sum_{i=\acute{t}}^{t} \mathbb{1}\left(\lambda_{u,m}^{\text{UAV}}(i) = \lambda_u(\acute{t})\right) d_{u,m}^{\text{com}}(i) = t$$

$$\text{or} \sum_{m\in\mathcal{M}} \sum_{i=\acute{t}}^{t} \sum_{j=i}^{t} \mathbb{1}\left(\lambda_u(\acute{t}) = \lambda_m^{\text{relay}}(i)\right)\cdot$$
$$\left.\mathbb{1}\left(\lambda_m^{\text{relay}}(i) = \lambda_{m,h}^{\text{HAP}}(j)\right) d_{m,h}^{\text{HAP}}(j) = t\right].$$

Based on the definition of $\mathcal{T}_{u,t}^{*}$, user device u can obtain a set of rewards $\{r_u(\acute{t})\}$ for tasks $\lambda_u(\acute{t})$ associated with time slot $\acute{t} \in \mathcal{T}_{u,t}^{*}$ at the beginning of time slot $t+1$. Then, device u stores the experience $(\boldsymbol{o}_u(\acute{t}), \boldsymbol{a}_u(\acute{t}), r_u(\acute{t}), \boldsymbol{o}_u(\acute{t}+1))$.

In traditional DRL, a mini-batch of experience transitions is randomly selected from the replay memory. However, the transitions are replayed with the same frequency regardless of their significance. In this approach, the negative influence of erroneous actions in the corresponding states cannot be learned rapidly by the agent. Hence, it is inefficient to replay all experience transitions uniformly. To solve the problems above, PER methods are proposed to select more useful experience transitions with higher probabilities [41], [42]. Both successful and failed attempts can generate useful experiences since successful attempts can provide the agent with positive rewards, and failed attempts are beneficial for preventing the agent from performing erroneous actions frequently. Hence, PER methods can enable agents to learn more rapidly and effectively from certain transitions, including successful and failed attempts.

The replay probability of each experience depends on the priority value. The absolute temporal-difference (TD) error is commonly utilized to measure the priority values of experiences. Let $\mathcal{I}$ denote a mini-batch of samples from memory R_u. For an experience $e_u^i = (\boldsymbol{o}_u(i), \boldsymbol{a}_u(i), r_u(i), \boldsymbol{o}_u(i+1)) \in \mathcal{I}$, the TD-error function is defined as

$$\text{TD}(e_u^i) = Q_u^{'}(i) - Q_u(\boldsymbol{o}_u(i), \boldsymbol{a}_u(i)|\boldsymbol{\theta}_u), \tag{31}$$

where $Q_u^{'}(i)$ denotes the target Q-value for experience e_u^i. The expression of $Q_u^{'}(i)$ is formulated as

$$Q_u^{'}(i) = r_u(i) + \gamma Q_u(\boldsymbol{o}_u(i+1), \boldsymbol{a}_u^{'}(i)|\boldsymbol{\theta}_u^{'}), \tag{32}$$

where $\boldsymbol{a}_u^{'}(i)$ represents the action with the maximum Q-value in the state $\boldsymbol{o}_u(i+1)$, i.e.,

$$\boldsymbol{a}_u^{'}(i) = \arg\max_{\boldsymbol{a}\in\mathbb{A}_u} Q_u\left(\boldsymbol{o}_u(i+1), \boldsymbol{a}|\,\boldsymbol{\theta}_u\right). \tag{33}$$

However, greedy TD-error prioritization leads to several issues. Some transitions with a low TD error may not be replayed during the long time after their first visit. In addition, it is sensitive to the circumstances in which the obtained rewards are stochastic. Finally, greedy prioritization may lead to over-fitting since it focuses only on a small subset of the replay memory. To overcome these issues, a stochastic sampling method is used to ensure a nonzero sampling probability for the lowest-priority experience. Specifically, the probability of sampling experience e_u^i is defined as

$$\text{prob}(e_u^i) = \frac{\left(\text{prio}(e_u^i)\right)^{\alpha}}{\sum_{e_u^i\in R_u}\left(\text{prio}(e_u^i)\right)^{\alpha}}, \tag{34}$$

where α represents a positive constant regarding the level of the prioritization, with $\alpha = 0$ corresponding to the uniform sampling. The notation $\text{prio}(e_u^i) > 0$ denotes the priority of the experience e_u^i, which is calculated by $\text{prio}(e_u^i) = 1/\text{rank}(e_u^i)$. Here, $\text{rank}(e_u^i)$ is the order of experience e_u^i when all experiences in the replay memory R_u are sorted according to the absolute TD error. In addition, an importance-sampling weight is introduced to correct the bias caused by the prioritized experience replay. We define the weight $\varpi(e_u^i)$ as

$$\varpi(e_u^i) = \left(\frac{1}{|R_u|\cdot\text{prob}(e_u^i)}\right)^{\iota}, \tag{35}$$

where $\iota \geq 1$ denotes an annealing variable. Algorithm 1 displays the training process of the agent on device $u \in \mathcal{U}$. Steps 11-17 show the details of the PER method. Steps 18-23 update the weights of the evaluate network and target network.

Algorithm 1: Training process of the agent u

```
1  Initialize: An initial replay memory $R_u$ for device $u$,
     the counter Num $= 0$, $\Delta = 0$;
2  for $e = 1$ to $E$ do
3      for $t \in \mathcal{T}$ do
4          if a new task $\lambda_u(t)$ arrives at device $u$ then
5              Take action $\boldsymbol{a}_u(t)$ based on (30);
6          end
7          Obtain the state $\boldsymbol{o}_u(t+1)$ and a set of rewards
             $\{r_u(\kappa), \kappa \in \mathcal{T}_{u,t}^{*}\}$;
8          for task $\lambda_u(\kappa)$ with $\kappa \in \mathcal{T}_{u,t}^{*}$ do
9              Store experience $e_u^{\kappa} = (\boldsymbol{o}_u(\kappa), \boldsymbol{a}_u(\kappa), r_u(\kappa),$
                 $\boldsymbol{o}_u(\kappa+1))$ in memory $R_u$ with maximal
                 priority $\text{prio}(e_u^{\kappa}) = \max_{i<\kappa}\text{prio}(e_u^i)$;
10         end
11         for $i \in \mathcal{I}$ do
12             Sample experience $e_u^i \sim \text{prob}(e_u^i)$;
13             Compute importance-sampling weight
                 $\varpi(e_u^i)$ according to (35);
14             Compute TD loss $\text{TD}(e_u^i)$ according to (31);
15             Update the replay probability $\text{prob}(e_u^i)$
                 according to (34);
16             Accumulate weight change $\Delta = \Delta +$
                 $\varpi(e_u^i)\cdot\text{TD}(e_u^i)\cdot\nabla_{\theta_u}Q_u(\boldsymbol{o}_u(i), \boldsymbol{a}_u(i)|\boldsymbol{\theta}_u)$;
17         end
18         Update weights $\boldsymbol{\theta}_u = \boldsymbol{\theta}_u + \eta\cdot\Delta$;
19         Reset $\Delta = 0$;
20         Num $=$ Num $+ 1$;
21         if mod (Num, Replace_Period) $= 0$ then
22             $\boldsymbol{\theta}_u^{'} = \boldsymbol{\theta}_u$;
23         end
24     end
25 end
```

5 Performance Evaluation

In this section, we evaluate the performance of our proposed computation offloading algorithm. First, the simulation setup is described. Then, we analyze the convergence and effectiveness of the algorithm. Finally, we compare the proposed algorithm with four benchmark methods in terms of the ratio of the dropped tasks and average cost.

5.1 Simulation Setup

We conduct simulations in the following scenario: one HAP at an altitude of 20 km and nine UAVs at an altitude of 2 km above the area with a size of 14×14 km^2. The coordinate

TABLE 2
Simulation Parameters

Parameters	Value
$\beta_u, \forall u \in \mathcal{U}$	$0.2 \sim 0.3$ Gigacycles/Mbits
$f_u, \forall u \in \mathcal{U}$	15 Gigacycles/s
$f_m, \forall m \in \mathcal{M}$	240 Gigacycles/s
f_h	45 Gigacycles/s
$P_u, \forall u \in \mathcal{U}$	0.5 W
$P_m, \forall m \in \mathcal{M}$	10 W
$B_u, \forall u \in \mathcal{U}$	5 MHz
$B_m, \forall m \in \mathcal{M}$	20 MHz
k_B	1.38×10^{-23} J/K
T_s	1000 K
N_0	-114 dBm
δ	0.1 s
T	1000

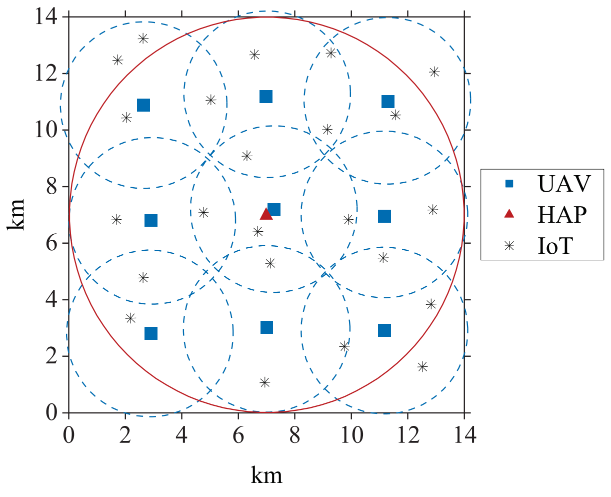


Fig. 5. The locations of the HAP and UAVs in the simulation scenario.

of the HAP is at the center of the area, and the UAVs are uniformly distributed in the area. An illustration of the simulation scenario is shown in Figure 5. Note that all the UAVs are within the coverage of the HAP, and all the terrestrial devices are within the coverage of the UAVs. The data size generated by each device is randomly selected in the range [20, 30] Mbits, and the maximum tolerable delay is 3 seconds. The experimental parameters and the corresponding values are shown in Table 2.

The python library TensorFlow (version 1.4.0) is utilized to construct ConvLSTM network and DQN. All the experiments are tested on a Windows workstation (CPU: Intel i7-7700 @3.6 GHz, RAM: 32 GB, GPU: NVIDIA GeForce GTX 1080). A 2-layer ConvLSTM with a convolution kernel size equal to 2×2 is adopted by the HAP to predict the task loads of all UAVs. The learning rate of the DQN is set to 0.001, and the discount factor is equal to 0.9. The replay buffer size is set to 500 and the batch size for training is set to 8. We provide six benchmark methods for experimental comparisons: 1) the random computation offloading method; 2) the branch-and-bound-based computation offloading method [14]; 3) the metaheuristic-based computation offloading method [43]; 4) the DQN-based computation offloading method; 5) the ConvLSTM-DQN-based computation offloading method; and 6) the PER-DQN-based computation offloading method.

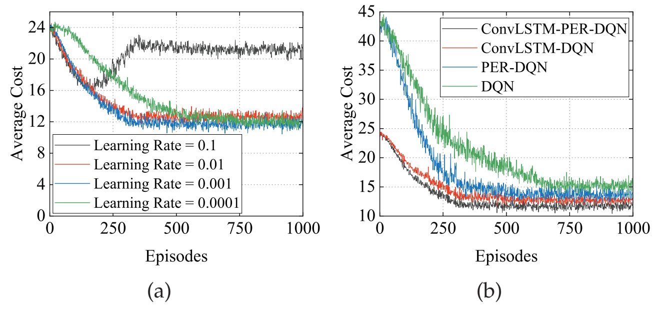


Fig. 6. The convergence performance of computation offloading methods. (a) The average cost vs. different learning rates. (b) The average cost vs. different methods during training.

TABLE 3
Overhead of DRL Model Training and Inference

Methods	TC	Params
DQN	900.12 s	3716 B
PER-DQN	610.31 s	6880 B
ConvLSTM-DQN	419.89 s	13400 B
ConvLSTM-PER-DQN	454.73 s	17280 B

5.2 Simulation Results

We study the impact of the learning rate on the convergence performance of our proposed algorithm, as shown in Figure 6(a). In the training process of ConvLSTM-PER-DQN, a learning rate of 0.1 is relatively high, and in the early stages of training, it might lead to overshooting or divergence. The model parameters may update too aggressively, causing the policy to become unstable and leading to an increase in reward (i.e., a decrease in average cost). Initially, these updates could improve the policy, but as the training progresses, large updates might cause the model to overshoot the optimal policy, leading to a decrease in performance. In addition, a high learning rate can lead to oscillations or instability in the optimization process. The model may struggle to find a stable policy due to large parameter updates, resulting in a fluctuation in reward. Reducing the learning rate can allow the model to stabilize and fine-tune its parameters more precisely, leading to an eventual increase in reward. In this paper, we adjust the learning rate in the set {0.1, 0.01, 0.001, 0.0001} to find an optimal learning rate that can stabilize the training process. When the learning rate is 0.0001, the convergence speed is slow. When the learning rate is 0.1, the agent cannot converge to a global optimum since a large learning rate causes the agent to become trapped into a local optimum. Therefore, when the learning rate is 0.001, the convergence speed is relatively fast, and the obtained average cost is the smallest.

Figure 6(b) illustrates the average cost of the computation offloading algorithms based on ConvLSTM-PER-DQN, ConvLSTM-DQN, PER-DQN, and DQN during the training

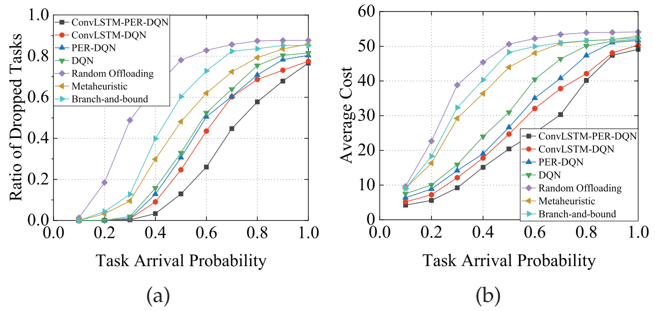


Fig. 7. The performance with different task arrival probabilities. (a) Ratio of dropped tasks. (b) Average cost.

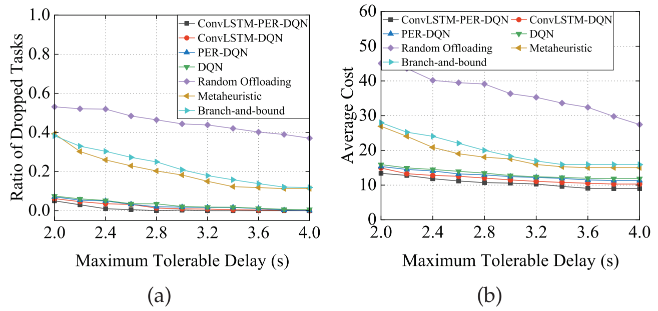


Fig. 8. The performance with different maximum tolerable delays. (a) Ratio of dropped tasks. (b) Average cost.

process. Initially, the average costs of ConvLSTM-PER-DQN and ConvLSTM-DQN are much lower than those of PER-DQN and DQN. This is because the ConvLSTM module can effectively extract dependencies from the sequential task loads of UAVs, which is capable of improving the training effectiveness. As the episode progresses, these computation offloading algorithms tend to be steady after 340, 500, 600, and 750 iterations. In addition, utilizing the PER method to train the agent can reduce the fluctuation of the average cost in the training process. Obviously, our proposed ConvLSTM-PER-DQN algorithm converges faster and achieves a lower average cost than the other three algorithms.

Table 3 lists the training and inference overhead of the four DQN-based computation offloading algorithms. We execute all the experiments on a Windows workstation. Due to the limitations of the experimental equipment, the training and inference overhead of the DQN and ConvLSTM network are considered together. The execution time of our proposed ConvLSTM-PER-DQN-based computation offloading algorithm is observably shorter than that of the other algorithms, while the overall parameter size of the ConvLSTM-PER-DQN-based algorithm is 4.6 times greater than that of the simplest DQN-based algorithm. This is because the parameter size of the ConvLSTM network occupies most of the storage space and is insignificant for HAP. With the advances in lightweight composite materials and autonomous avionics, several HAPs, such as STRATOBUS [44] and Elevate [45], which are able to stay floated in the stratosphere with more than 100 kg of payload, can provide enough storage capacity and computational capacity for training the ConvLSTM network. In addition, if some elementary IoT devices cannot afford the overhead of DRL model training, the training task can be offloaded to the MEC-mounted UAV or MEC-mounted HAP, and the IoT devices only need to receive the network parameters that have already been trained.

In Figure 7, we investigate the correlation between the task arrival probability and the ratio of dropped tasks as well as the average cost. Figure 7(a) shows that the four DRL-based computation offloading algorithms yield a lower ratio of dropped tasks than do the branch-and-bound-based algorithm and metaheuristic-based algorithm as the task arrival probability increases. It is worth noting that our proposed computation offloading algorithm based on ConvLSTM-PER-DQN outperforms the other three DQN-based algorithms, which reveals that the combination of ConvLSTM, PER and DQN is particularly effective. When the task arrival probability is high (i.e., 0.8), the ConvLSTM-PER-DQN- based algorithm can always keep the ratio of dropped tasks less than 0.6, while the ratios of dropped tasks obtained by other computation offloading algorithms increase to more than 0.7. Figure 7(b) shows the average cost as a function of the task arrival probability. As the task arrival probability increases, the average cost gradually increases since more tasks are generated in the system. When the task arrival probability increases from 0.2 to 0.5, the growth rate of the average cost of the ConvLSTM-PER-DQN algorithm is slower than that of the other algorithms. This finding implies that the growth in the average cost of our proposed computation offloading algorithm is the mildest. When the task arrival probability is 0.5, the ConvLSTM-PER-DQN-based algorithm outperforms the other algorithms by at least 14.8% shorter average cost. As the task arrival probability reaches 0.8, the ConvLSTM-PER-DQN algorithm outperforms the other algorithms by at least 4.1% shorter average cost.

Figure 8(a) and Figure 8(b) investigate the effect of the maximum tolerable delay on the ratio of dropped tasks and the average cost. As illustrated in Figure 8(a), the ConvLSTM-PER-DQN-based computation offloading algorithm perpetually obtains a lower ratio of dropped tasks than do the other baseline algorithms, particularly when the maximum tolerable delay is small. This finding implies that the ConvLSTM-PER-DQN-based algorithm achieves better performance for delay-sensitive tasks. When the maximum tolerable delay is 2.0 s, the ConvLSTM-PER-DQN-based algorithm decreases the ratio of dropped tasks by 18.5%∼33% compared with the other three DQN-based algorithms. In addition, the ConvLSTM-PER-DQN- based algorithm decreases the ratio of dropped tasks by more than 86% as compared to the branch-and-bound algorithm and metaheuristic algorithm. With the increase in the maximum tolerable delay, the ratio of dropped tasks of all algorithms gradually decreases. As the deadline of each task reaches 3.2 s, the ratio of dropped tasks of our proposed ConvLSTM-PER-DQN algorithm is zero and ultimately becomes steady. This is because when the maximum tolerable delay is long, the system has enough time to transmit and process the generated tasks. Figure 8(b) shows that the average cost of each algorithm gradually decreases and finally becomes steady when the maximum tolerable delay is prolonged.

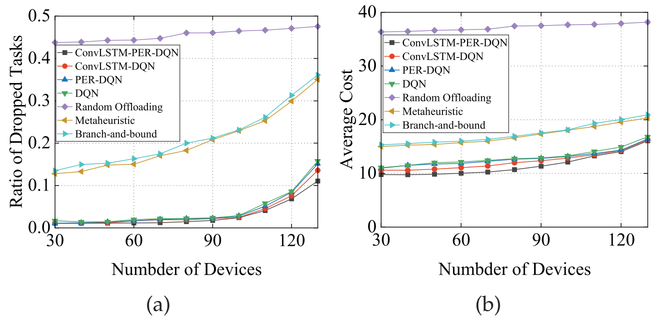


Fig. 9. The performance with different numbers of devices. (a) Ratio of dropped tasks. (b) Average cost.

This is because when the deadline is extended, tasks that require long processing time and transmission time can be executed completely before the deadline, and the penalty will not be counted in the system cost. When the deadline is large enough (i.e., 3.2 s), there are no dropped tasks and no differences in the average cost with further increase in the maximum tolerable delay. The converged average cost of the ConvLSTM-PER-DQN-based algorithm is approximately 9.01.

In Figure 9, we investigate how the number of devices affects the ratio of dropped tasks and average cost. As shown in Figure 9(a), the branch-and-bound-based algorithm and metaheuristic-based algorithm have a large space as compared with the four DRL-based computation offloading algorithms. In addition, a lower ratio of dropped tasks can be achieved by the ConvLSTM-PER-DQN-based computation offloading algorithm than by other algorithms, particularly in the case of a large number of devices. As the number of devices reaches 100, the ratio of dropped tasks of the ConvLSTM-PER-DQN algorithm is less than 0.02. This is because ConvLSTM can effectively react to the dynamic task loads of UAVs. In Figure 9(b), the average cost of each algorithm increases with the increase in the number of devices. This is because an increasing number of tasks are processed, and their processing costs are counted in the system cost. When the number of devices increases to 130, the average cost of the ConvLSTM-PER-DQN algorithm is approximately 16, and the ConvLSTM-PER-DQN algorithm achieves an average cost of 20.9% and 23.2% lower than those of the metaheuristic algorithm and branch-and-bound algorithm, respectively.

6 Conclusion

In this paper, we develop an aerial hierarchical MEC system by cooperating MEC-mounted HAPs and MEC-mounted UAVs. A non-divisible computing task has three choices: 1) computed locally; 2) offloaded to a UAV and computed by the UAV; or 3) offloaded to the HAP through a UAV relay and computed by the HAP. We formulate a problem of minimizing the total processing cost, which is intractable to solve by conventional optimization approaches. Hence we propose a multi-agent DRL-based computation offloading algorithm that incorporates the ConvLSTM network and PER methods. The ConvLSTM network is utilized to predict the task loads of UAVs, and the PER method is beneficial for accelerating convergence during the model training. The experimental results show that our proposed computation offloading algorithm outperforms the baselines with respect to the average cost and the ratio of dropped tasks.

References

[1] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, Sept. 2019.

[2] Z. Zhao, R. Zhao, J. Xia, X. Lei, D. Li, C. Yuen, and L. Fan, "A novel framework of three-hierarchical offloading optimization for MEC in industrial IoT networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5424–5434, Aug. 2020.

[3] B. Ji, Y. Wang, K. Song, C. Li, H. Wen, V. G. Menon, and S. Mumtaz, "A survey of computational intelligence for 6G: Key technologies, applications and trends," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 10, pp. 7145–7154, Oct. 2021.

[4] J. Qiu, D. Grace, G. Ding, M. D. Zakaria, and Q. Wu, "Air-ground heterogeneous networks for 5G and beyond via integrating high and low altitude platforms," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 140–148, Dec. 2019.

[5] M. S. Alam *et al.*, "High altitude platform station based super macro base station constellations," *IEEE Communications Magazine*, vol. 59, no. 1, pp. 103–109, Jan. 2021.

[6] M. A. Knodler Jr, C. D. Fitzpatrick, D. Ni, M. Plotnikov, L. Fiondella, W. S. Mogawer, D. Chen, T. Chigan, Y. Xie *et al.*, "The application of unmanned aerial systems in surface transportation-volume I: Executive summary," University of Massachusetts at Amherst. Transportation Center, Tech. Rep., Dec. 2019.

[7] S. H. Alsamhi, F. Almalki, O. Ma, M. S. Ansari, and B. Lee, "Predictive estimation of optimal signal strength from drones over IoT frameworks in smart cities," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 402–416, Apr. 2021.

[8] S. H. Alsamhi, F. Afghah, R. Sahal, A. Hawbani, M. A. Al-qaness, B. Lee, and M. Guizani, "Green internet of things using UAVs in B5G networks: A review of applications and strategies," *Ad Hoc Networks*, vol. 117, p. 102505, Jun. 2021.

[9] "HAPSMobile." [Online], Available: https://www.hapsmobile.com/.

[10] P. Serrano, M. Gramaglia, F. Mancini, L. Chiaraviglio, and G. Bianchi, "Balloons in the sky: unveiling the characteristics and trade-offs of the Google loon service," *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3165–3178, Dec. 2021.

[11] N. Cheng, W. Xu, W. Shi, Y. Zhou, N. Lu, H. Zhou, and X. Shen, "Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 26–32, Aug. 2018.

[12] G. K. Kurt, M. G. Khoshkholgh, S. Alfattani, A. Ibrahim, T. S. Darwish, M. S. Alam, H. Yanikomeroglu, and A. Yongacoglu, "A vision and framework for the high altitude platform station (HAPS) networks of the future," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 729–779, Mar. 2021.

[13] J. Wu, H. Lin, H. Liu, and L. Gao, "A deep reinforcement learning approach for collaborative mobile edge computing," in *IEEE International Conference on Communications*, Seoul, Korea, May 2022, pp. 601–606.

[14] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[15] K. Li, "Heuristic computation offloading algorithms for mobile users in fog computing," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 2, pp. 1–28, Jan. 2021.

[16] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. Honolulu, HI, USA: IEEE, Oct. 2018, pp. 207–215.

[17] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, Apr. 2020.

[18] H. Zhou, Z. Zhang, D. Li, and Z. Su, "Joint optimization of computing offloading and service caching in edge computing-based smart grid," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1122–1132, Apr. 2022.

[19] G. Zhang, S. Zhang, W. Zhang, Z. Shen, and L. Wang, "Joint service caching, computation offloading and resource allocation in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 20, no. 8, pp. 5288–5300, Mar. 2021.

[20] L. Wang and G. Zhang, "Joint service caching, resource allocation and computation offloading in three-tier cooperative mobile edge computing system," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3343–3353, Mar. 2023.

[21] M. Bolourian and H. Shah-Mansouri, "Energy-efficient task offloading for three-tier wireless powered mobile edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 400–10 412, Jan. 2023.

[22] H. Peng and X. Shen, "Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 131–141, Jan. 2021.

[23] W. Zhou, L. Fan, F. Zhou, F. Li, X. Lei, W. Xu, and A. Nallanathan, "Priority-aware resource scheduling for UAV-mounted mobile edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 7, pp. 9682–9687, Feb. 2023.

[24] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6960, Sept. 2022.

[25] W. Lu, Y. Mo, Y. Feng, Y. Gao, N. Zhao, Y. Wu, and A. Nallanathan, "Secure transmission for multi-UAV-assisted mobile edge computing based on reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 3, pp. 1270–1282, Jun. 2023.

[26] 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. 18 725–18 736, Oct. 2022.

[27] Y. Liu, J. Yan, and X. Zhao, "Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime UAV communication network," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 4225–4236, Apr. 2022.

[28] Z. Wang, H. Rong, H. Jiang, Z. Xiao, and F. Zeng, "A load-balanced and energy-efficient navigation scheme for UAV-mounted mobile edge computing," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3659–3674, Jul. 2022.

[29] M. Ke, Z. Gao, Y. Huang, G. Ding, D. W. K. Ng, Q. Wu, and J. Zhang, "An edge computing paradigm for massive IoT connectivity over high-altitude platform networks," *IEEE Wireless Communications*, vol. 28, no. 5, pp. 102–109, Oct. 2021.

[30] S. Wang, M. Chen, C. Yin, W. Saad, C. S. Hong, S. Cui, and H. V. Poor, "Federated learning for task and resource allocation in wireless high-altitude balloon networks," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 460–17 475, Dec. 2021.

[31] Y. Yang, X. Chang, Z. Jia, Z. Han, and Z. Han, "Towards 6G joint HAPS-MEC-cloud 3C resource allocation for delay-aware computation offloading," in *IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking*, Exeter, United Kingdom, Jun. 2020, pp. 175–182.

[32] Q. Ren, O. Abbasi, G. K. Kurt, H. Yanikomeroglu, and J. Chen, "Caching and computation offloading in high altitude platform station (HAPS) assisted intelligent transportation systems," *IEEE Transactions on Wireless Communications*, vol. 21, no. 11, pp. 9010–9024, Nov. 2022.

[33] C. Ding, J.-B. Wang, H. Zhang, M. Lin, and G. Y. Li, "Joint optimization of transmission and computation resources for satellite and high altitude platform assisted edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 2, pp. 1362–1377, Aug. 2021.

[34] Z. Jia, Q. Wu, C. Dong, C. Yuen, and Z. Han, "Hierarchical aerial computing for internet of things via cooperation of HAPs and UAVs," *IEEE Internet of Things Journal*, vol. 10, no. 7, pp. 5676–5688, Apr. 2023.

[35] S. Mao, S. He, and J. Wu, "Joint UAV position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3992–4002, Sept. 2021.

[36] H. Liao, Z. Zhou, X. Zhao, and Y. Wang, "Learning-based queue-aware task offloading and resource allocation for space–air–ground-integrated power IoT," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5250–5263, Apr. 2021.

[37] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-UAV-enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6898–6908, Aug. 2020.

[38] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *Journal of combinatorial optimization*, vol. 4, pp. 171–186, Jun. 2000.

[39] A. R. Pathak, M. Pandey, and S. Rautaray, "Application of deep learning for object detection," *Procedia computer science*, vol. 132, pp. 1706–1717, Jun. 2018.

[40] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019.

[41] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine learning*, vol. 13, pp. 103–130, Oct. 1993.

[42] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, San Juan, Puerto Rico, USA, May 2016, pp. 1–21.

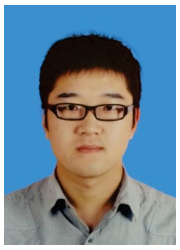
[43] J. Wang, T. Liu, K. Liu, B. Kim, J. Xie, and Z. Han, "Computation offloading over fog and cloud using multi-dimensional multiple knapsack problem," in *IEEE Global Communications Conference*. Abu Dhabi, United Arab Emirates: IEEE, Feb. 2018, pp. 1–7.

[44] "What's up with stratobus?" [Online], Available: https://www.thalesgroup.com/en/worldwide/space/news/whats-stratobus.

[45] "Elevate." [Online], Available: http://www.zero2infinity.space/elevate/.

Yuanyuan Wang received the B.E. degree from HeFei University of Technology, China, in 2017. She is currently pursuing the Ph.D. degree with the University of Science and Technology of China, Hefei. Her research interests include wireless ad hoc network, network optimization, and edge computing.

Chi Zhang received the B.E. and M.E. degrees in electrical and information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1999 and 2002, respectively, andthe Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, USA, in 2011. In 2011, he joined the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, as an Associate Professor. His current research interests include network protocol design and performance analysis and network security, particularly for wireless networks and social networks.



Taiheng Ge received the B.E. degree from An-Hui University of Technology, China, in 2013. He is currently pursuing the Ph.D. degree with the University of Science and Technology of China, Hefei. Her research interests include wireless communications and network optimization.

Miao Pan received the B.Sc. degree in electrical engineering from the Dalian University of Technology, China, in 2004, the M.A.Sc. degree in electrical and computer engineering from the Beijing University of Posts and Telecommunications, China, in 2007, and the Ph.D. degree in electrical and computer engineering from the University of Florida in 2012. He was an Assistant Professor in computer science with Texas Southern University from 2012 to 2015. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Houston. His research interests include cognitive radio networks, cyber-physical systems, and cybersecurity. He received best paper awards in Globecom 2017 and Globecom 2015, respectively. He is a member of IEEE. He is currently an Associate Editor of the IEEE INTERNET OF THINGS JOURNAL.