Graphical Abstract

DRL-based Task Offloading for Networked UAVs with Random Mobility and Collision Avoidance[⋆]

Xixin Zhang, Dongge Jia, Junfei Xie

Highlights

DRL-based Task Offloading for Networked UAVs with Random Mobility and Collision Avoidance

Xixin Zhang, Dongge Jia, Junfei Xie

- Research highlight 1
- Research highlight 2

DRL-based Task Offloading for Networked UAVs with Random Mobility and Collision Avoidance

Xixin Zhang[a,b,1], Dongge Jia[b,2], Junfei Xie[b,3,*]

[a]*Department of Electrical and Computer Engineering, University of California San Diego, 9500 Gilman Dr, La Jolla, 92092, California, USA*
[b]*Department of Electrical and Computer Engineering, San Diego State University, 5500 Campanile Drive, San Diego, 92182, California, USA*

Abstract

Unmanned Aerial Vehicles (UAVs) have gained widespread use across various fields due to their flexibility and multifunctionality. However, their limited onboard computing capacity is often criticized for hindering their ability to execute complex tasks in real-time. To address this challenge, Networked Airborne Computing (NAC) has emerged, which leverages the collective computing power of multiple UAVs to enable efficient handling of large-scale data processing, real-time analytics, and complex mission coordination. Despite its potential, research in this area is still in its infancy. In this paper, we consider a typical NAC scenario where multiple UAVs with collision avoidance capabilities share resources while moving randomly within an area. Without prior knowledge of the system models, we aim to optimize task allocation among UAVs with uncertain mobility. To achieve this, we propose a Deep Reinforcement Learning algorithm based on the Twin Delayed Deep Deterministic Policy Gradient (TD3). Simulation results demonstrate that our approach significantly speeds up task execution compared to existing meth-

[⋆]This document is the results of the research project funded by the National Science Foundation.

[*]Corresponding author

Email addresses: `xiz166@ucsd.edu` (Xixin Zhang), `xxxx@email.edu` (Dongge Jia), `jxie4@sdsu.edu` (Junfei Xie)

[1]This is the first author footnote.

[2]Another author footnote, this is a very long footnote and it should be a long footnote. But this footnote is not yet sufficiently long enough to make two lines of footnote text.

[3]Yet another author footnote.

ods.

Keywords: Computation Offloading, Deep Reinforcement Learning, Unmanned Aerial Vehicle, Edge Computing

1. Introduction

In recent years, unmanned aerial vehicles (UAVs), or drones, have seen rapid advancements and growing popularity in areas such as precision agriculture, disaster response, aerial photography, and environmental monitoring [1, 2]. As UAV applications become increasingly complex, the use of multiple cooperative UAVs has become more common. Nevertheless, their limited onboard computational resources often become a bottleneck. One solution that naturally follows is to offload computationally intensive tasks to external resources.

Extensive research has focused on efficiently utilizing resources on edge servers or remote clouds to support multi-UAV applications. For instance, Liu *et al*. [3] proposed to utilize a UAV-Edge-Cloud computing model and formulate a joint optimization of workflow assignment and multi-hop routing scheduling for UAV swarms to minimize computation cost and latency. Bai *et al*. [4] investigated delay-aware cooperative task offloading for multi-UAV enabled edge-cloud computing, proposing an algorithm to balance task distribution and minimize completion delay. In these studies, UAVs in the swarm are typically viewed as relays that bring edge servers or remote clouds closer, rather than computing nodes. They get sufficient computing resources at the cost of a high data transmission delay, which may not be acceptable for time-sensitive UAV applications not to mention real-time tasks. Moreover, mobile edge servers require a reliable local network infrastructure, which is difficult to deploy and scale, especially in underdeveloped or post-disaster areas [5].

With technological advancements, the emergence of small, lightweight yet powerful micro-computers has significantly accelerated the onboard computing capacity of UAVs. This has spurred researchers to explore UAVs' potential in acting as edge servers. In [6], Hu *et al*. leveraged the computing resources of a moving UAV to serve ground users, aimed to minimize the total maximum delays among users by jointly optimizing offloading ratios, user scheduling, and UAV trajectory in a UAV-aided mobile edge computing system. Miao *et al*. [7] proposed a multi-UAV-assisted mobile edge computing (MEC) offloading algorithm that maximizes the access quantity and minimizes the task completion latency by cluster path planning based on user mobility and communication coverage. Although UAVs have proven promising in providing on-demand computing resources, these studies treat them as separate servers.

To harness the full computational potential of multi-UAV systems, a new paradigm called Networked Airborne Computing (NAC) is proposed, where multiple aerial vehicles share resources among each other[8]. The fast deployment, infrastructure-free, and low-cost characteristics make the UAV-based NAC a promising technique. Nevertheless, research in NAC is still in its early stages. In our previous studies, we have developed a ROS-based simulator and a hardware testbed that consists of multiple UAVs to facilitate NAC research [9]. In [10], we introduced a coded distributed computing scheme based on deep reinforcement learning (DRL) for optimally partitioning and allocating tasks to multiple networked UAVs. This scheme addresses two typical NAC scenarios. The first scenario involves uncontrollable UAV mobility, which can happen when they are operated by different owners. In the second scenario, UAVs are controlled to assist in task computation. Simulation results demonstrate the effectiveness of the proposed scheme. However, in the first scenario, we assumed UAVs maintain a consistent movement pattern throughout the execution of a particular task and did not account for motion interference between UAVs due to collision avoidance. Moreover, the simple matrix multiplication tasks were considered.

In this paper, we investigate a more common yet challenging NAC scenario where all UAVs, including both offloaders and offloadees, move randomly during task execution while actively avoiding collisions. None of the UAVs have prior knowledge of the environment or system models, and their movement patterns or future trajectories are not shared among each other. Additionally, we generalize computation tasks as any functions or operations that can be partitioned into arbitrary subtasks for parallel computation. To model UAV movement, we extend the traditional Random Direction model [11], originally designed for individual entities, to capture collision avoidance interactions among multiple UAVs. Furthermore, we formulate a nonlinear optimization to optimize task allocation and develop a DRL algorithm based on the Twin Delayed Deep Deterministic Policy Gradient (TD3)[12] to solve it. We evaluate the performance of the proposed method through extensive comparative simulation studies, which demonstrate its promising performance.

In the rest of this paper, Sec. 2 details the system models and formulates the optimization problem. Sec. 3 describes the proposed DRL algorithm. In Sec. 4, simulation results are presented and discussed. We conclude in Sec. **??**.

2. System Models and Problem Formulation

In this section, we first introduce the system models used to construct the simulated environment, which are unknown to the UAVs. Then we formulate the problem to be solved.

2.1. System Models

Consider a group of $N+1$ heterogeneous UAVs with varying physical configurations, indexed as $i \in \mathcal{N} = \{0, 1, 2, ..., N\}$. Each UAV is equipped with computing and communication modules, enabling resource sharing and onboard computation. Their computing, communication, and mobility characteristics can be modeled as follows.

2.1.1. Computing Model

We describe the computing capability of each UAV i as CPU cycle frequency f_i in Hz. For a general computing task k, its input data size is S_k (bits), and its required computation intensity is ξ_k (cycles/bit)[13]. The total CPU cycles required to compute task k is hence $\xi_k S_k$ and the time required for UAV i to execute this task is

$$T_{k,i}^{comp} = \frac{\xi_k S_k}{f_i} \tag{1}$$

2.1.2. Communication Model

Denote the distance between UAV i and UAV j as d_{ij}. The UAV-to-UAV links are typically Line of Sight (LoS), with propagation speed approaching the speed of light. Hence, the transmission latency can be approximated using the transmission time. Here, we model the transmission rate (bits/s) based on the Simplified Path Loss Model [14] as follows

$$\nu_{ij} = B \log_2(1 + \frac{G(d_r/d_{ij})^{\theta}\psi_i}{N_0 B}) \tag{2}$$

where B is the bandwidth (Hz) of the channel, d_r is the reference distance (meter), G is the unitless constant equal to the path gain of the distance d_r,

θ is the path loss exponent, ψ_i is the transmitted power (mW) and N_0 is the noise power spectral density (dBm/Hz). The overall transmission time is as follows

$$T_{k,ij}^{trans} = \frac{S_k}{\nu_{ij}} \tag{3}$$

2.1.3. Mobility Model

We assume the UAVs fly at the same altitude. Therefore, the position of each UAV i at time t can be depicted as $\mathbf{p}_i(t) = (x_i(t), y_i(t)) \in \mathbb{R}^2$ with constraints $0 \leq x_i(t) \leq W$, $0 \leq y_i(t) \leq W$, such that the position is bounded within an area of $W \times W (m^2)$. To model its movement, we adopt the Random Direction (RD) model [11], which has been widely used for describing UAVs, particularly multirotor drones [15]. Given initial position $\mathbf{p}_i(0) = (x_i(0), y_i(0))$, UAV i randomly picks a constant velocity, where the magnitude ranges uniformly between 0 and $v_{max} \in \mathbb{R}$, and the direction is uniformly distributed across 2π. The UAV then moves in a straight line for a duration randomly selected from an exponential distribution with parameter λ. Once this duration is completed, the UAV chooses another velocity and duration, repeating the process. Suppose at the m-th instance when UAV i changes its velocity, it selects a new velocity $\mathbf{v}_{i,m} \in \mathbb{R}^2$ and duration $\delta_{i,m} \in \mathbb{R}$. We define the start time of the m-th instance as $t_m = \sum_{l=-1}^{m-1} \delta_{i,l}$, with $\delta_{i,-1} = 0$. The UAV i's position at time t, where $t_m \leq t < t_{m+1}$, can be represented as follows

$$\mathbf{p}_i(t) = \mathbf{p}_i(0) + \sum_{l=-1}^{m-1} \delta_{i,l}\mathbf{v}_{i,l} + (t - t_m)\mathbf{v}_{i,m} \tag{4}$$

The traditional RD model is designed to describe the mobility of a single entity. However, in multi-UAV systems, the mobility of UAVs can change to avoid collisions. This necessitates the incorporation of collision avoidance mechanisms into the RD model. Here, we assume that each UAV i will turn around and move in the opposite direction without changing speed until the current duration is completed when its distance to any other UAV j falls below a threshold D_i. Note that if both UAVs have the same threshold $D_i = D_j$, UAV j will also reserve its direction. By treating the boundaries of the area as obstacles, we ensure that all UAVs move within the designated area. The mobility of each UAV i with collision avoidance can then be described as

$$\begin{aligned}\mathbf{p}_i(t) = \mathbf{p}_i(0) + \sum_{l=-1}^{m-1} (\delta^o_{i,l} - \delta^c_{i,l})\mathbf{v}_{i,l} \\ + (\tilde{\delta}^o_{i,m} - \tilde{\delta}^c_{i,m})\mathbf{v}_{i,m}\end{aligned} \tag{5}$$

where $0 \leq \delta^o_{i,l} \leq \delta_{i,l}$ is the time spent moving at velocity $\mathbf{v}_{i,l}$ in the l-th instance before triggering collision avoidance and $\delta^c_{i,l} = \delta_{i,l} - \delta^o_{i,l}$. Likewise, $0 \leq \tilde{\delta}^o_{i,m} \leq t - t_m$ is the time spent moving at velocity $\mathbf{v}_{i,m}$ before collision avoidance in the m-th instance, and $\tilde{\delta}^c_m = (t - t_m) - \tilde{\delta}^o_m$.

2.2. Problem Formulation

Without loss of generality, we let UAV $i = 0$ be the master (or offloader) and treat the remaining N UAVs as potential offloadees with idle computing resources. Suppose a sequence of computing tasks $\mathcal{K} = \{1, 2, ..., K\}$ is generated at the master, and each task k can be divided into $L_k \in \mathbb{Z}^+$ atomic tasks of size $\ell_k = \frac{S_k}{L_k}$, which can be computed in parallel. To minimize the total task completion time, the master aims to optimally allocate the L_k atomic tasks among all the UAVs, including itself. Suppose the workload offloaded to UAV $i \in \mathcal{N}$ for task k is $c^k_i \ell_k$, where c^k_i is a non-negative integer that satisfies $0 \leq c^k_i \leq L_k$. Therefore, $\sum_{i=0}^{N} c^k_i = L_k$. Of note, when there is no workload offloaded to UAV i, then $c^k_i = 0$.

Let T_k denote the time taken to compute task k, and $T_{k,i}$ represent the time taken by UAV i to receive the data, process it and return the result back to the master. We then have

$$T_k = \max_i \; T_{k,i} \tag{6}$$

For simplicity, we assume the result size is small and the latency of sending it back is negligible, as often assumed in existing works [16]. Therefore, $T_{k,i}$ can be expressed as

$$T_{k,i} = T^{comp}_{k,i} + T^{trans}_{k,i} \tag{7}$$

According to the computing and communication models described in the previous section, we can derive that $T^{comp}_{k,i} = \frac{\xi_k c^k_i \ell_k}{f_i}$ and $T^{trans}_{k,i}$ satisfies

$$\begin{cases} \int_0^{T^{trans}_{k,i}} \nu_{0i}(t)\, dt = c^k_i \ell_k & \text{if } i \neq 0 \\ T^{trans}_{k,i} = 0 & \text{if } i = 0 \end{cases} \tag{8}$$

The problem can then be mathematically formulated as follows

$$\underset{c_i^k, \forall i \in \mathcal{N}, k \in \mathcal{K}}{\text{Minimize}} \quad \sum_{k=1}^{K} T_k \tag{9a}$$

$$\text{subject to} \quad \sum_{i=0}^{N} c_i^k \ell_k = S_k \quad \forall k \in \mathcal{K} \tag{9b}$$

$$c_i^k \in \mathbb{Z},\, 0 \leq c_i^k \leq L_k, \forall i \in \mathcal{N}, k \in \mathcal{K} \tag{9c}$$

3. Deep Reinforcement Learning Solution

To solve the optimization problem formulated in the previous section, the greatest challenge lies in the unknown relationship between T_k and the decision variables c_i^k, as UAVs lack knowledge of the system models. Additionally, the randomness of UAVs' mobility presents another significant challenge. In this section, we introduce our model-free DRL-based algorithm, a variant of the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [12], to address these challenges.

3.1. Markov Decision Process

We first covert the optimization problem into a Markov Decision Process represented by a tuple $(\mathcal{S}, \mathcal{A}, r, P)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the state transition model, and r is the reward function. The master UAV is the DRL agent that takes actions to minimize the total task completion time.

3.1.1. State

We assume the master agent only has access to all UAVs' positions $\mathbf{p}_i(t), \forall i \in \mathcal{N}$ at each time t, and the prior knowledge about their configurations, *i.e.* idle computing resources f_i, and communication capabilities characterized by transmitted powers ψ_i. We discretize the time into steps of length Δt. Let t^k denote the start time of task k, the state at t^k is defined as the combination of the task size S_k, all UAVs' historic trajectories and configurations $s_k = [S_k, (\boldsymbol{\rho}_0(k), f_0, \psi_0), (\boldsymbol{\rho}_1(k), f_1, \psi_1), ..., (\boldsymbol{\rho}_N(k), f_N, \psi_N)] \in \mathcal{S}$, where $\boldsymbol{\rho}_i(k) = [\mathbf{p}_i(t^k - M\Delta t), \mathbf{p}_i(t^k - (M-1)\Delta t), ..., \mathbf{p}_i(t^k)] \in \mathbb{R}^{2\times(M+1)}$ is UAV i's trajectory consisting of its most recent M-step positions (including the start point).

3.1.2. Action

Every time a task k arrives, the master agent partitions it into sub-tasks of varying amounts of atomic tasks and offloads them to different UAVs. Let $\mu_i^k \geq 0$ denote the portion of task k offloaded to UAV $i \in \mathcal{N}$, such that $\sum_{i=0}^{N} \mu_i^k = 1$. The action taken by the master at time t^k is defined as $a_k = [\mu_0^k, \mu_1^k, ..., \mu_N^k] \in \mathcal{A}$, where $\mathcal{A}$ is the space of N-simplex. The workload offloaded to UAV i is then computed by $c_i^k = \text{round}(\frac{\mu_i^k S_k}{\ell_k})$.

3.1.3. Reward

To minimize the total task completion time, we define the reward function as follows

$$r(s_k, a_k) = \frac{S_k}{\sum_{k \in \mathcal{K}} S_k} \cdot \frac{\tilde{T}_k - T_k}{\tilde{T}_k} \tag{10}$$

where $\tilde{T}_k = \frac{\xi_k S_k}{f_0}$ represents the time required to execute task k locally at the master and $\frac{T_k - \tilde{T}_k}{T_k}$ indicates the acceleration rate achieved by offloading the task to nearby UAVs. $\frac{S_k}{\sum_{k \in \mathcal{K}} S_k}$ is the weight of task k among all tasks.

3.1.4. Transition

As discussed in Sec. 2, all UAVs move randomly according to the random mobility model with collision avoidance schemes. Hence, the transition model, which describes the state transitions given the current action, depends on the random mobility model and can be abstracted as follows

$$s_{k+1} \sim P(s_{k+1} | s_k, a_k; v_{max}, \lambda) \tag{11}$$

where $P(\cdot)$ represents the transition model, whose explicit form is unknown. v_{max} and λ are parameters of the random mobility model.

3.2. TD3-based Offloading

TD3 [12] is an advanced actor-critic algorithm designed for continuous action spaces in DRL. Here, we introduce a variant of the TD3 algorithm to enable the master UAV to identify trustworthy offloadees and optimize task allocation.

3.2.1. Actor

The behavior of the master agent is defined by a policy function $\pi : \mathcal{S} \rightarrow \mathcal{A}$, which maps each state $s \in \mathcal{S}$ to a continuous action $a \in \mathcal{A}$. The goal of the agent is to determine the optimal policy π^*, which maximizes the expected return defined as $J = \mathbb{E}_{s_k \sim P, a_k \sim \pi}[R_1]$, where $R_k = \sum_{i=k}^{K} \gamma^{i-k} r(s_i, a_i)$ is the accumulative discounted reward and γ is the discount factor.

To approximate the policy function, TD3 [12] utilizes a neural network parameterized by ϕ, denoted as π_ϕ, which directly maps the state space $s \in \mathcal{S}$ to the action space $\mathcal{A}$. To satisfy the constraints in (9b), we adjust the actor π_ϕ as follows. By including all UAVs' historic trajectories in the state, the neural network can learn the movement patterns of each UAV and their interactions. Moreover, the network adjusts weights to prioritize UAVs that have more resources to share and are more likely to remain close to the master. Therefore, the neural network's output logits $\mathbf{z}^k \in \mathbb{R}^{N+1}$ can be interpreted as the reliability score of each UAV. We then use the Softmax function to decide the offloading portions according to UAVs' reliability scores as follows

$$\mu_i^k = \text{Softmax}(z_i^k) = \frac{e^{z_i^k}}{\sum_{j=0}^{N} e^{z_j^k}} \tag{12}$$

where z_i^k represents the reliability score of UAV i for completing task k.

To estimate the parameters ϕ, we apply off-policy learning to enhance training stability. Moreover, to strengthen the robustness of the learned policy function against variance and prevent overfitting to narrow peaks of action values, we add random noise to the actions of the target actor $\pi_{\phi'}$ parameterized by ϕ' as follows [12]

$$\begin{aligned} \tilde{\mu}_i^k =& (1-\beta)\mu_i^k + \beta\epsilon_i \\ \epsilon \sim& \ \text{Dir}(\alpha_{policy}) \end{aligned} \tag{13}$$

where β is the weight of noise ϵ_i, and $\epsilon_i \in \mathbb{R}^{N+1}$ is sampled from the Dirichlet distribution [17] with a concentration parameter α_{policy} that is uniform across all elements. Of note, the Dirichlet distribution guarantees that the actions remain within the space of N-simplex.

3.2.2. Critic

Given the state s_k and the action a_k taken, the state-action value function Q^π provides the expected return when following policy π thereafter, i.e.,

$Q^{\pi}(s_k, a_k) = \mathbb{E}_{s_{i>k}\sim P, a_{i>k}\sim\pi}[R_k|s_k, a_k]$. To approximate the critic Q^{π}, neural networks are also used. Following TD3[12], we define two primary critic networks Q_{θ_1} and Q_{θ_2} and two target critic networks $Q_{\theta'_1}$ and $Q_{\theta'_2}$ parameterized by θ_1, θ_2, θ'_1 and θ'_2, respectively.

3.2.3. Training

As shown in Alg. 1, the training starts by initializing all the parameters and the replay buffer $\mathcal{D}$, which stores transition samples (Lines 1-3). At each training iteration u, the master agent interacts with the environment to collect transition data and store them in the buffer $\mathcal{D}$ until the number of collected transitions exceeds H (Lines 5-6). After that, the agent utilizes a batch $\mathcal{B}$ sampled from the replay buffer to update the parameters of the critics (Lines 9-11) and actor (Lines 12-13). Particularly, the parameters θ_i, $i \in \{1, 2\}$, of each primary critic is updated by minimizing the following loss over the sampled batch

$$l_i = \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s')\in\mathcal{B}} (y - Q_{\theta_i}(s, a))^2 \tag{14}$$

where $y = r(s, a) + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a}')$ and $\tilde{a}' = (\tilde{\mu}_0, \tilde{\mu}_1, ..., \tilde{\mu}_N)$ is the regularized action defined in (13).

A lower update frequency is necessary for the policy function compared to the value function, otherwise, it may lead to divergence. Hence, we update the policy function every d iterations by maximizing the expected return in the direction of the batch gradient of the policy, where the gradient is computed by

$$\nabla_{\phi} J = \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s')\in\mathcal{B}} [\nabla_a Q_{\theta_1}(s, a)|_{a=\pi_{\phi}(s)} \nabla_{\phi}\pi_{\phi}(s)] \tag{15}$$

Also, the parameters of the target networks are gradually adjusted towards the weights of the primary networks through weighted soft updates every d iteration as follows

$$\begin{aligned} \theta'_i &\leftarrow \theta_i + (1-\tau)(\theta'_i - \theta_i), i = 1, 2 \\ \phi' &\leftarrow \phi + (1-\tau)(\phi' - \phi) \end{aligned} \tag{16}$$

where $\tau \in [0, 1]$ is the weight.

Algorithm 1 TD3 training for task offloading

```
1: Initialize the critic networks $Q_{\theta_1}, Q_{\theta_2}$ by randomly assigning values to $\theta_1, \theta_2$, and initialize the target critic networks $Q_{\theta'_1}, Q_{\theta'_2}$ by setting $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2$
2: Initialize the actor network $\pi_\phi$ and the target actor $\pi_{\phi'}$ by randomly assigning values to $\phi$ and setting $\phi' \leftarrow \phi$
3: Initialize the replay buffer by $\mathcal{D} \leftarrow \emptyset$
4: for $u = 1$ to $U$ do
5:     Select action $a \sim (1-\beta)\pi_\phi(s) + \beta\epsilon$, with exploration noise $\epsilon \sim \mathrm{Dir}(\alpha_{explore})$, observe reward $r$ and new state $s'$
6:     Store the transition tuple $(s, a, r, s')$ in $\mathcal{D}$
7:     if $u > H$ then
8:         Sample a batch of $|\mathcal{B}|$ transitions $(s, a, r, s')$ from buffer $\mathcal{D}$
9:         $\tilde{a}' \leftarrow (1-\beta)\pi_{\phi'}(s') + \beta\epsilon,\ \epsilon \sim \mathrm{Dir}(\alpha_{policy})$
10:        $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a}')$
11:        Update the critics by $\theta_i \leftarrow \arg\min_{\theta_i} \frac{1}{|\mathcal{B}|} \sum (y - Q_{\theta_i}(s,a))^2$, $i = 1, 2$
12:        if $u \mod d = 0$ then
13:            Update the actor by $\phi \leftarrow \arg\max_\phi \frac{1}{|\mathcal{B}|} \sum Q_{\theta_1}(s, \pi_\phi(s))$
14:            Update the target network by $\theta'_i \leftarrow \theta_i + (1-\tau)(\theta'_i - \theta_i), i = 1, 2$ and $\phi' \leftarrow \phi + (1-\tau)(\phi' - \phi)$
15:        end if
16:    end if
17: end for
```

4. Experiments

In this section, we conduct experiments to evaluate the effectiveness and scalability of our proposed TD3-based DRL algorithm.

4.1. Environment Setting

We evaluate our method in simulated scenarios with one master UAV and $N = 3$, 6, 9, or 12 offloadee UAVs respectively. For the multi-UAV RD mobility model, we set its parameters as $v_{max} = 20$ m/s, $\lambda = \frac{1}{15}$, and $D_i = 5$, $\forall i \in \mathcal{N}$. The length of each discrete time step is set to $\Delta t = 1$ second. We also vary the size of the flying zone and consider two sizes, $W = 300$m and $W = 400$m. The computing power f_i of each UAV i is randomly configured by selecting values from the range of $[1, 1.6]$ Ghz. The task list consists of $K = 25$ tasks that can be locally completed in $\tilde{T}_k \in [20, 60]$ seconds by the master UAV. All tasks can be divided into $L_k = 1000$ atomic tasks. The computation intensity is set to be the same $\xi_k = 10^6$ cycles/kB for all tasks, and the size of each task is determined by $S_k = \frac{f_0 \tilde{T}_k}{\xi_k}$. As a case of a networked UAV swarm utilizing 5G Wi-Fi communication for data transmission, we set the bandwidth $B = 40$ MHz, relative distance $d_r = 1$ meter, path gain $G = 40$, path loss exponent $\theta = 4$, and noise power spectral density $N_0 = -174$ dBm/Hz. The transmitted power ψ_i of the master UAV to each UAV $i \in \mathcal{N} \setminus \{0\}$ varies from 80 mW to 120 mW.

4.2. Training Performance

We employ a 3-layer Multilayer Perceptrons (MLP)[18] architecture for both the actor and critic networks. The actor network takes observations as input, whose dimension is based on the number of computing nodes $N + 1$, and outputs actions of dimension N. Each UAV trajectory has a length of $M + 1$, where $M = 20$. The critic network takes the concatenation of observations and actions as input and outputs a scalar representing the state action value. The width of the hidden layer in both networks is set to twice the dimension of the input. All parameters are initialized using Kaiming initialization[19].

The learning rates for the actor and critic networks are set to 0.0001 and 0.0002, respectively. The threshold H is set to 1000 and the batch size is $|\mathcal{B}| = 512$. The gradient norm is clipped between 0 and 0.2. The exploration and policy noise are sampled from a Dirichlet distribution with parameters $\alpha_{explore} = 0.1$ and $\alpha_{policy} = 0.99$, respectively. The noise weight is $\beta = 0.1$

and the discount factor γ is 0.99. The actor network is updated every $d = 25$ iterations, and the soft update weight τ for target networks is 0.005.

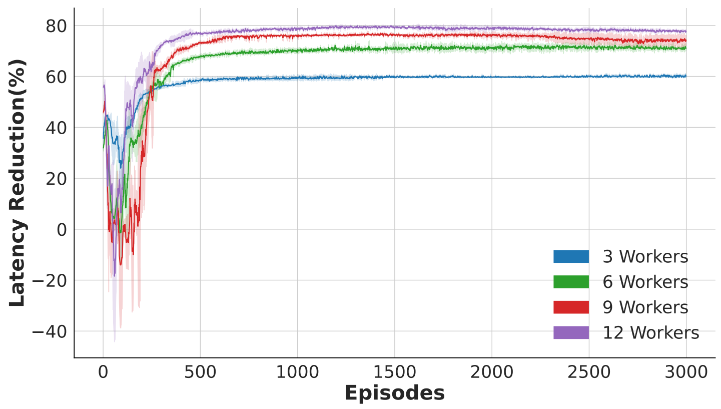

Figure 1: Learning Curve ($300 \times 300m^2$)

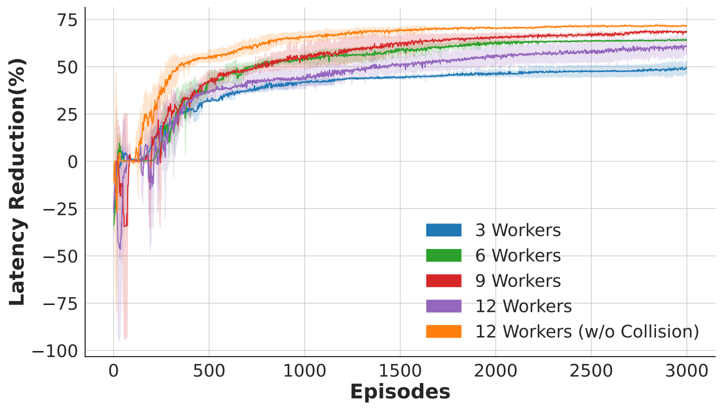

Figure 2: Learning Curve($400 \times 400m^2$)

In each scenario, we train the agent for 3000 episodes, i.e., $U = 3000K = 75000$ iterations, with three different random seeds. The results are shown in Fig. 1 and Fig. 2. The latency reduction is defined as the reduction in total task completion by task offloading compared to local computing at the master UAV, i.e.,

$$\text{Latency Reduction} = \frac{\sum_{k\in\mathcal{K}} \tilde{T}_k - \sum_{k\in\mathcal{K}} T_k}{\sum_{k\in\mathcal{K}} \tilde{T}_k} \times 100\%.$$

The two figures demonstrate that our method converges after training and shows promising stability. When UAVs are restricted to an area of $300 \times 300m^2$ (Fig. 1), increasing the number of potential offloadees (workers) reduces task completion time. The same phenomenon is observed when the flying zone expands to $400 \times 400m^2$, except when the number of potential workers increases to $N = 12$. The performance degradation at $N = 12$ may be due to increased collision avoidance maneuvers, which make UAVs' mobility more uncertain and harder to learn and predict. Intuitively, the master agent tends to share workload with UAVs that are more likely to remain nearby throughout task execution, as indicated by a higher reliability score z_i^k. Therefore, when UAV mobility becomes more uncertain, fewer workers are selected as offloadees due to reduced reliability. This is confirmed by the results shown in Fig. 2, where performance improves when collision avoidance mechanisms are not in place. It also indicates that the task completion time cannot be infinitely reduced by continuously adding more UAVs to the region.

4.3. Comparison Studies

To evaluate the performance of the proposed method, we compare it with five benchmarks, including (1) **Equal (all)** that equally divides and distributes tasks to all UAVs; (2) **Equal (close)** that equally partitions and distributes tasks to UAVs that are close enough with distance to the master satisfying $d_i <$ 100m; (3) **Naive Prediction** that selects offloadees based on predicted future trajectories and assigns sub-tasks of equal size to these offloadees. Specifically, it predicts each UAV's trajectory for the next 20 steps, assuming the UAVs maintain their current velocity and ignoring potential collisions. It then calculates the percentage q_i of predicted UAV positions that remain within 200m of the master ($d_i <$ 200m). UAVs with $q_i \geq 40\%$ are selected as offloadees; (4) **Reliable** that allocates tasks based on a reliability score defined as reliability $= q_i \psi_i f_i$. It picks the top $\lceil \frac{N+1}{2} \rceil$ UAVs with the highest reliability scores and assigns tasks to these UAVs proportionally to their reliability scores; (5) **Random** that randomly samples actions from the action space.

The results in Fig. 3 and Fig. 4 demonstrate the promising performance of our method, which achieves the highest latency reduction across all scenarios. When the area is $300 \times 300m^2$, the **Naive Prediction** ranks second in scenarios with 3, 6, and 9 workers, while the **Equal (close)** ranks second in scenarios with 12 workers. When the area expands to $400 \times 400m^2$ (Fig.

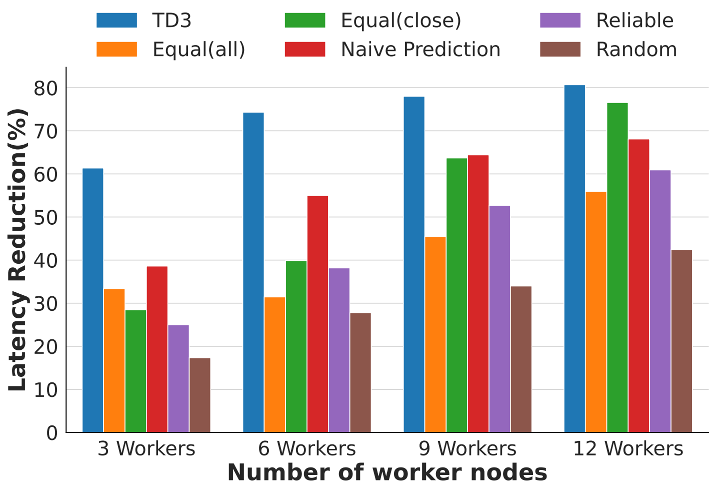

Figure 3: Performance Comparison ($300 \times 300m^2$)

4), the **Equal (all)** and the **Random** provide no benefit in reducing latency; instead, they significantly delay task completion. Comparing Fig. 3 and Fig. 4, we can observe that as the area increases for a fixed N, the performance of all methods degrades. This is due to the sparser airspace, which causes UAVs to be farther apart from each other, thereby increasing transmission latency and task completion time.

Appendix A. Example Appendix Section

Appendix text.

Example citation, See [4].

References

[1] H. Kurunathan, H. Huang, K. Li, W. Ni, E. Hossain, Machine learning-aided operations and communications of unmanned aerial vehicles: A contemporary survey, IEEE Communications Surveys & Tutorials (2023).

[2] Y. Zeng, R. Zhang, T. J. Lim, Wireless communications with unmanned aerial vehicles: Opportunities and challenges, IEEE Communications magazine 54 (5) (2016) 36–42.

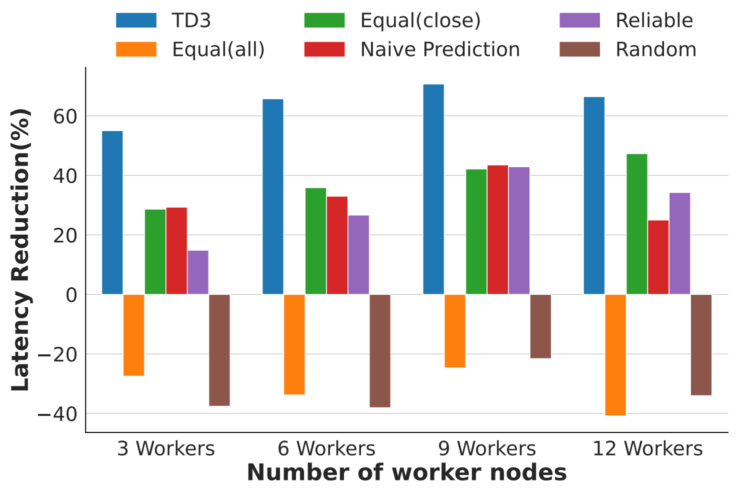

Figure 4: Performance Comparison ($400 \times 400m^2$)

[3] B. Liu, W. Zhang, W. Chen, H. Huang, S. Guo, Online computation offloading and traffic routing for uav swarms in edge-cloud computing, IEEE Transactions on Vehicular Technology 69 (8) (2020) 8777–8791.

[4] Z. Bai, Y. Lin, Y. Cao, W. Wang, Delay-aware cooperative task offloading for multi-uav enabled edge-cloud computing, IEEE Transactions on Mobile Computing (2022).

[5] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39.

[6] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, G. Y. Li, Joint offloading and trajectory design for uav-enabled mobile edge computing systems, IEEE Internet of Things Journal 6 (2) (2018) 1879–1892.

[7] Y. Miao, K. Hwang, D. Wu, Y. Hao, M. Chen, Drone swarm path planning for mobile edge computing in industrial internet of things, IEEE Transactions on Industrial Informatics (2022).

[8] K. Lu, J. Xie, Y. Wan, S. Fu, Toward uav-based airborne computing, IEEE Wireless Communications 26 (6) (2019) 172–179.

[9] H. Zhang, B. Wang, R. Wu, J. Xie, Y. Wan, S. Fu, K. Lu, Exploring networked airborne computing: A comprehensive approach with advanced simulator and hardware testbed, Unmanned Systems (2023).

[10] B. Wang, J. Xie, K. Lu, Y. Wan, S. Fu, Learning and batch-processing based coded computation with mobility awareness for networked airborne computing, IEEE Transactions on Vehicular Technology (2022).

[11] E. M. Royer, P. M. Melliar-Smith, L. E. Moser, An analysis of the optimum node density for ad hoc mobile networks, in: ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No. 01CH37240), Vol. 3, IEEE, 2001, pp. 857–861.

[12] S. Fujimoto, H. van Hoof, D. Meger, Addressing Function Approximation Error in Actor-Critic Methods (Oct. 2018). arXiv:1802.09477.

[13] K. Cheng, Y. Teng, W. Sun, A. Liu, X. Wang, Energy-efficient joint offloading and wireless resource allocation strategy in multi-mec server systems, in: 2018 IEEE international conference on communications (ICC), IEEE, 2018, pp. 1–6.

[14] A. Goldsmith, Wireless communications, Cambridge university press, 2005.

[15] D. S. Lakew, U. Sa'ad, N.-N. Dao, W. Na, S. Cho, Routing in flying ad hoc networks: A comprehensive survey, IEEE Communications Surveys & Tutorials 22 (2) (2020) 1071–1120.

[16] N. T. Hoa, N. C. Luong, D. Van Le, D. Niyato, et al., Deep reinforcement learning for multi-hop offloading in uav-assisted edge computing, IEEE Transactions on Vehicular Technology (2023).

[17] C. M. Bishop, N. M. Nasrabadi, Pattern recognition and machine learning, Vol. 4, Springer, 2006.

[18] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, nature 323 (6088) (1986) 533–536.

[19] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.