electronics

MDPI

Article

Learning-Based Collaborative Computation Offloading in UAV-Assisted Multi-Access Edge Computing

Zikun Xu [1,†], Junhui Liu [1,2,†], Ying Guo [1], Yunyun Dong [1,3] and Zhenli He [1,2,3,*]

[1] School of Software, Yunnan University, Kunming 650504, China; xuzikun@stu.ynu.edu.cn (Z.X.); hanks@ynu.edu.cn (J.L.); guoy@mail.ynu.edu.cn (Y.G.); dongyy929@ynu.edu.cn (Y.D.)
[2] Yunnan Key Laboratory of Software Engineering, Yunnan University, Kunming 650504, China
[3] Engineering Research Center of Cyberspace, Yunnan University, Kunming 650504, China
* Correspondence: hezl@ynu.edu.cn
† These authors contributed equally to this work.

Abstract: Unmanned aerial vehicles (UAVs) have gained considerable attention in the research community due to their exceptional agility, maneuverability, and potential applications in fields like surveillance, multi-access edge computing (MEC), and various other domains. However, efficiently providing computation offloading services for concurrent Internet of Things devices (IOTDs) remains a significant challenge for UAVs due to their limited computing and communication capabilities. Consequently, optimizing and managing the constrained computing, communication, and energy resources of UAVs are essential for establishing an efficient aerial network infrastructure. To address this challenge, we investigate the collaborative computation offloading optimization problem in a UAV-assisted MEC environment comprising multiple UAVs and multiple IODTs. Our primary objective is to obtain efficient offloading strategies within a multi-heterogeneous UAV environment characterized by limited computing and communication capabilities. In this context, we model the problem as a multi-agent markov decision process (MAMDP) to account for environmental dynamics. We employ a multi-agent deep deterministic policy gradient (MADDPG) approach for task offloading. Subsequently, we conduct simulations to evaluate the efficiency of our proposed offloading scheme. The results highlight significant improvements achieved by the proposed offloading strategy, including a notable increase in the system completion rate and a significant reduction in the average energy consumption of the system.

Keywords: UAV-assisted MEC; multi-agent reinforcement learning; task offloading; multi-agent deep deterministic policy gradient

Citation: Xu, Z.; Liu, J.; Guo, Y.; Dong, Y.; He, Z. Learning-Based Collaborative Computation Offloading in UAV-Assisted Multi-Access Edge Computing. *Electronics* **2023**, *12*, 4371. https://doi.org/10.3390/electronics12204371

Academic Editor: Carlo Mastroianni

Received: 26 September 2023
Revised: 18 October 2023
Accepted: 19 October 2023
Published: 22 October 2023

1. Introduction

A variety of computationally and communication-intensive applications, such as virtual reality, facial recognition, and autonomous driving, are anticipated to operate on resource-constrained Internet of Things devices (IOTDs) [1,2]. Simultaneously, IOTDs introduce a spectrum of requirements encompassing diverse service quality levels and computational resources. Nevertheless, the limited computational resources pose a significant impediment to the widespread adoption of these innovative applications. Multi-access edge computing (MEC) emerges as an effective solution to tackle this challenge. By offloading computational tasks from IOTDs with restricted computing capabilities and finite battery power to edge servers located in close proximity, MEC has the potential to significantly reduce both latency and energy consumption.

However, deploying edge servers in remote or underdeveloped terrestrial areas presents significant challenges. For instance, establishing edge servers in sparsely populated regions like isolated mountains and islands may not be a cost-effective proposition. In contrast, the UAV-assisted MEC environment offers a promising solution to meet the

needs of underserved regions where terrestrial base stations are either absent or insufficient [3,4].

The UAV-assisted MEC environment harbors substantial potential in future wireless systems, especially for maritime applications and emergency scenarios. This technology can extend coverage to IOTDs in remote locations, hotspots, and emergency zones that may have limited access to terrestrial base stations or are underserved by conventional means [5–7]. Researchers are increasingly drawn to the UAV-assisted MEC environment due to its adaptability and cost-effectiveness. A significant challenge in deploying the UAV-assisted MEC environment lies in efficiently offloading a variety of tasks from diverse IOTDs. This offloading should be based on considerations such as the computational resources and energy consumption of the UAVs.

Numerous researchers have made significant contributions to the field of UAV-assisted MEC environments, addressing various aspects such as energy efficiency [8], trajectory planning [9], and the joint optimization of communication and computation [10]. However, these existing research findings come with certain limitations. Firstly, a majority of current scenarios center around a single UAV, necessitating the formation of UAV swarms to distribute the computational workload due to the constrained processing capacity of individual UAVs. Secondly, many approaches tackle computation offloading problems using traditional optimization techniques, often as one-time optimizations that consider a specific system state. Given the dynamic nature of UAV-assisted MEC environments, effectively applying these methods proves to be challenging.

In response to this challenge, this paper conducts a comprehensive investigation into the optimization problem of cooperative computing offloading strategies within UAV-assisted MEC environments. The principal contributions of this study can be succinctly summarized as follows:

- We investigate the problem of collaborative computation offloading involving multiple UAVs under the constraints of limited communication range and latency. Our goal is to collectively minimize the average energy consumption while increasing the overall efficiency.
- We transform the optimization problem of cooperative computation offloading into a multi-agent markov decision process (MAMDP). In this framework, each UAV serves as an individual agent, and these agents make decisions by considering local real-time observations and the current policy. These decisions involve selecting the task offloading target and allocation ratio.
- We design an algorithm based on multi-agent deep deterministic policy gradient (MADDPG). With offline training, each UAV can make real-time offloading decisions. We conduct a series of simulation experiments to prove the effectiveness of the algorithm in dynamic environments.

This paper employs a multi-agent reinforcement learning approach to address the optimization of cooperative user computation offloading policies in a UAV-assisted MEC environment characterized by constrained communication ranges and computing resources. The remainder of the paper is arranged as follows. Section 2 reviews current literature relevant to our study. Section 3 introduces the mathematical model for the considered UAV-assisted MEC environment. Section 4 presents the solution based on the MADDPG algorithm. Section 5 conducts a series of simulation experiments to demonstrate the effectiveness of our proposed algorithm. Section 6 provides a summary of this paper.

2. Related Work

The computation offloading problem has consistently been a research focal point in the context of UAV-assisted MEC environments, with numerous research contributions of significant importance. To this end, we have conducted a comprehensive review of existing research and categorized it into three key domains: optimization-based offloading methods, game-theory-based offloading methods, and reinforcement-learning-based offloading methods.

Optimization-based offloading methods. Optimization-based offloading methods have found widespread application in task offloading within the UAV-assisted MEC environment. These algorithms primarily focus on single or multi-objective optimization to enhance performance. Diao et al. [11] introduced an alternating optimization algorithm that combines a greedy approach, successive convex approximation, and functional analysis methods. They applied this algorithm to jointly optimize offloading parameters and UAV trajectories with the goal of minimizing the average peak age of information, average energy consumption of IOTDs, and average energy consumption of UAVs. Lv et al. [12] employed Bayesian optimization to ascertain the optimal offloading decision, aiming to minimize overall energy consumption while maintaining adherence to mission quality of service requirements. Chen et al. [13] devised a parallel offloading strategy using Liapunov optimization. This strategy efficiently identifies optimal decisions for latency-sensitive and computation-intensive tasks, achieving joint optimization encompassing green energy utilization, task division factors, CPU frequencies, and transmission power. Hu et al. [14] introduced a three-step block coordinate descent algorithm. This algorithm concurrently optimizes the reflection coefficient of reconfigurable intelligent surfaces, the received beam formation vector at the access point, and the local energy allocation strategy for user equipment. The aim is to maximize the total completed task input bits for all users. Guo et al. [15] employed a Lyapunov-based approach to optimize multi-user partial computational offloading. Their goal was to minimize the energy consumption of all users while ensuring compliance with time delay constraints. Luo et al. [16] introduced a particle swarm optimization (PSO) algorithm for the joint optimization of offloading decisions, communication allocation, and computational resources. Their objective was to minimize both latency and cost. Abbas et al. [17] introduced a real-time computing task offloading and resource allocation strategy with a user-centric approach. They employed the Lyapunov drift plus penalty optimization method to create a dynamic partial offloading solution aimed at minimizing energy consumption and monetary costs while maximizing task completion rates. Sheng et al. [18] employed a joint optimization approach for offloading decisions and resource allocation, utilizing a convex-concave iterative process. This method effectively mitigates co-channel interference while addressing differentiated upload delays among users.

Game theory-based offloading methods. Game theory-based offloading methods have been successfully applied to formulate, design, or optimize operations in various representative environments [19]. Zhou et al. [20] introduced a novel optimization model designed to maximize the expected offloading rate of multiple agents. They achieved this by optimizing the offloading threshold and then transformed the problem into a game-theoretic framework. Their analysis included an examination of the existence of Nash equilibrium strategies within this game-theoretic model. Pham et al. [21] formulated the offloading decision problem as a potential game and devised a distributed offloading scheme with low complexity using a game-theoretic approach. Xiao et al. [22] addressed MEC grouping for task offloading in MEC cooperation by framing it as a utility maximization problem. They accomplished this by creating a non-cooperative game strategy choice using regret matching. Chen et al. [23] framed the multiuser task offloading problem as a game and derived a Nash equilibrium strategy for task offloading. Teng et al. [24] converted the multi-server multi-task allocation and scheduling problem into a non-cooperative game. They demonstrated the existence of a Nash equilibrium and introduced a low-complexity response updating algorithm that converges to the Nash equilibrium. Fang et al. [25] framed the problem of minimizing the overall latency for all users in the network as a matching game considering heterogeneous demands. They employed a multi-round cooperative matching algorithm to discover a stable match for this game. You et al. [26] introduced a novel game-theoretic algorithm for multilayer computational and network resource allocation in the context of computational offloading within the MEC environment. Zhang et al. [27] introduced a game-theory-based approach to determine the optimal

offloading strategy aimed at minimizing the weighted cost associated with time delay and energy consumption.

Reinforcement-learning-based offloading methods. Reinforcement learning methods have garnered substantial attention for resolving intricate decision problems in dynamic and uncertain environments [28]. Wang et al. [29] introduced a task offloading method based on meta-reinforcement learning. This approach facilitates rapid and adaptive task offloading decisions in MEC environments by learning the initial parameters of the task offloading policy. Liu et al. [30] presented a deep reinforcement learning-based approach for offloading and resource allocation to enhance system performance and efficiency. Zhou et al. [31] proposed a combined strategy for computational offloading and resource allocation utilizing deep reinforcement learning to decrease the energy consumption of the entire MEC system. Tang et al. [32] introduced a distributed algorithm, leveraging model-free deep reinforcement learning, to optimize offloading decisions with the goal of minimizing expected long-term costs. Dai et al. [33] presented an asynchronous actor–critic algorithm aimed at identifying the optimal stochastic computational offloading strategy, with the objective of minimizing long-term energy efficiency in industrial networks. Gao et al. [34] proposed a framework based on the proximal policy optimization algorithm. This framework addresses location privacy protection and facilitates effective task offloading decisions, achieving a balance between computational task performance requirements and user location privacy preservation. Wang et al. [35] achieved efficient computational offloading by employing intelligent decision-making strategies within a reinforcement learning framework. This approach is aimed at reducing the energy consumption and latency of mobile devices. Lu et al. [36] introduced a deep reinforcement learning-driven computation offloading approach that efficiently transfers computational tasks to edge servers. This approach relies on intelligent decision-making strategies to enhance computational performance and user experience.

To clearly distinguish our research and underscore its unique characteristics, we conducted a comparative analysis between our study and the previously mentioned works.

- First, we address the problem of optimizing cooperative offloading involving multiple UAVs. Considering the constraints of limited computing resource and communication range, our primary goal is to minimize the average energy consumption of these UAVs while ensuring latency constraints are met.
- Second, the predominant approaches in the current landscape heavily rely on game-theory-based methods and optimization-based techniques, which are deterministic optimization methods well-suited for static environments. However, their adaptability diminishes when faced with dynamic environmental changes and the necessity for continuous offloading. To address this dynamism and the continuous nature of offloading decisions, we incorporate deep reinforcement learning methods.
- Third, the majority of existing research predominantly relies on single-agent reinforcement learning algorithms, which makes it challenging to address cooperative issues involving multiple agents. To overcome this limitation, we enhance our approach by adopting the MADDPG framework, which transforms the multi-user cooperative offloading game into a multi-agent model.

3. Models and Problem Definition

In this section, we first formulate mathematical models to describe the UAV-assisted MEC environment. We then provide a formal definition of the optimization problem under consideration. The mathematical notations used in this paper are summarized in Abbreviations, where the symbols are listed in the order introduced in the paper.

3.1. The UAV-Assisted MEC Environment

In this paper, we consider a computing environment comprising n UAVs (denoted as $\mathrm{UAV}_1, \mathrm{UAV}_2, \ldots, \mathrm{UAV}_n$) in addition to multiple IOTDs. Each UAV is equipped with limited

computational and communication capabilities, serving as a task offloading provider for the IOTDs [8,37], as illustrated in Figure 1.

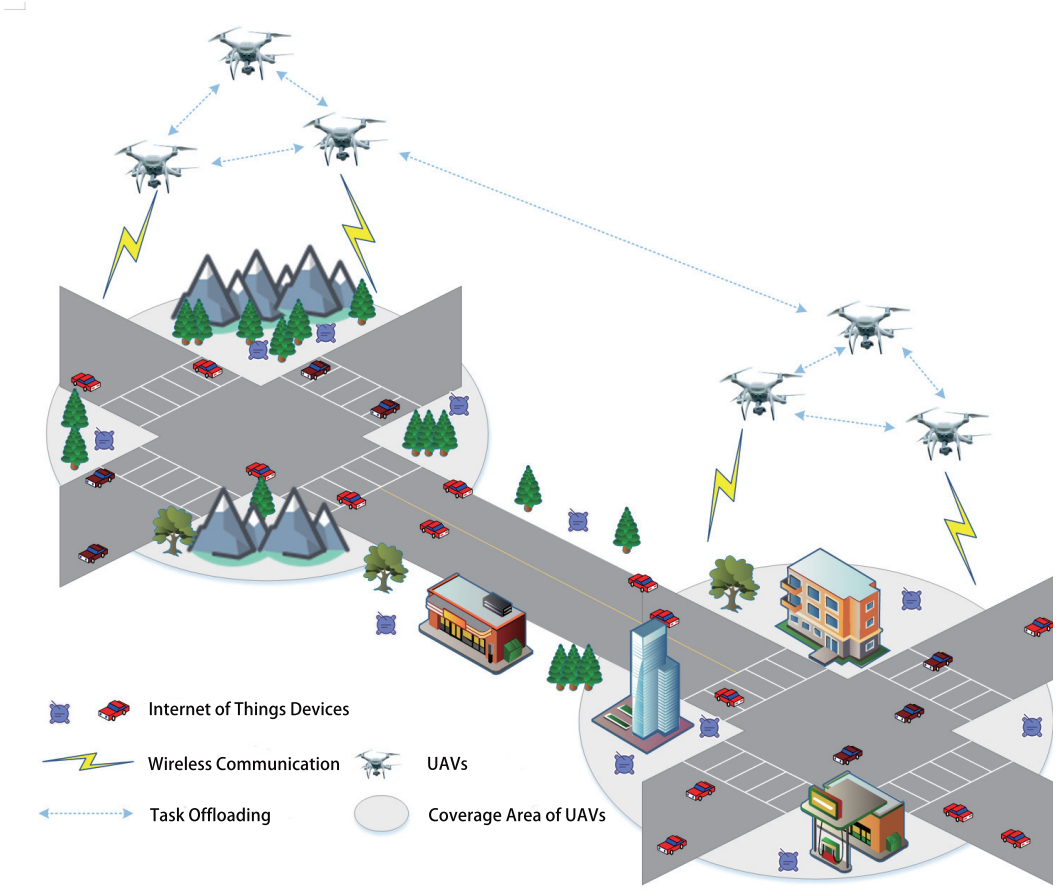


Figure 1. A UAV-assisted MEC environment with multiple UAVs and multiple IOTDs.

We assume that each UAV adheres to a predetermined flight path while operating independently. Positioned at fixed coordinates, a UAV can provide computation offloading services to IOTDs within its coverage area. Periodically, UAVs receive computationally intensive tasks of varying sizes from IOTDs. To ensure compliance with the latency constraint due to limited computational and communication capabilities, each UAV offloads tasks to other UAVs within its communication range. The main objective of this work is to minimize the average energy consumption while maintaining the average response latency constraint.

In this framework, we discretize the runtime into discrete time slots $t \in \{1, 2, 3, \ldots, T\}$, where T represents the finite time horizon. UAVs receive computationally intensive tasks from IOTDs at the initiation of each time slot. The UAVs tailor their offloading schemes based on the state information of other UAVs observed in the environment. UAV_i gauges the communication circumstances with other UAVs in the prevailing environment through multi-sensor information fusion techniques, denoted as $U_i = (\mathrm{UAV}_i^1, \mathrm{UAV}_i^2, \ldots, \mathrm{UAV}_i^n)$. $\mathrm{UAV}_i^j = 1$ signifies that UAV_j falls within the communication range of UAV_i; conversely, $\mathrm{UAV}_i^j = 0$, with $1 \leq j \neq i \leq n$ indicates the opposite. Considering the energy constraints of UAVs, it is essential that the energy consumed for computation and communication remains within the limits of the UAV's battery power. Addressing this, potential solutions, such as current charging methods [38,39], could prevent battery depletion.

3.2. Offloading Schemes of UAVs

Typically, in UAV-assisted MEC environments, tasks can be executed locally or offloaded to a ground base station or a cloud center for processing. However, we introduce

an innovative concept where tasks can also be offloaded to nearby UAVs for collaborative computation. The aim of this innovative approach is to maximize computational resource utilization across multiple UAVs, considering the limitations imposed by UAVs' limited communication range, with the objective of improving task execution efficiency and performance.

Let d_i^t (measured in units of bits, $1 \leq i \leq n$) denote the size of the offloaded computational tasks received by UAV_i from IOTDs at the beginning of the time slot t. To ensure the completion of these computational tasks within the predefined latency constraints, each UAV is required to formulate its own offloading strategy based on the information it gathers. That is, each UAV must determine the percentage of computational tasks to be offloaded to other UAVs within its communication range.

We use vector $\left(\lambda_{i,1}^t, \lambda_{i,2}^t, \ldots, \lambda_{i,n}^t\right)$ to denote the offloading scheme of UAV_i at time slot t. In this representation, $\lambda_{i,i}^t$ signifies the percentage of computational tasks that UAV_i handles locally, while $\lambda_{i,j}^t$ $(i \neq j)$ denotes the percentage of computational tasks that UAV_i offloads to UAV_j for collaborative execution. Then, we have $\lambda_{i,1}^t + \lambda_{i,2}^t + \cdots + \lambda_{i,n}^t = 1$. Note that the UAVs that are not in the communication range of UAV_i do not participate in the offloading decision process, i.e., if $\text{UAV}_i^j = 0$, then $\lambda_{i,j}^t = 0$.

Let us consider that the number of CPU cycles needed to execute 1-bit data on UAV_i is denoted as c_i (measured in cycles/bit). Since the size of data not offloaded from UAV_i to other UAVs for processing at time slot t is $\lambda_{i,i}^t d_i^t$ bits, the number of CPU cycles required to process this data is given by

$$C_i^{loc,t} = c_i \lambda_{i,i}^t d_i^t. \tag{1}$$

Furthermore, UAV_i also receives computational tasks from other UAVs at time slot t, resulting in a total data size of the received computational tasks of $\sum_{j=1,j\neq i}^{n} \lambda_{j,i}^t d_j^t$ bits. Consequently, the CPU cycles required to process these computational tasks from other UAVs on UAV_i can be expressed as

$$C_i^{acc,t} = \sum_{j=1,j\neq i}^{n} c_i \cdot \lambda_{j,i}^t d_j^t. \tag{2}$$

Therefore, the total number of CPU cycles required for all computational tasks to be executed by UAV_i at time slot t can be expressed as

$$C_i^t = C_i^{loc,t} + C_i^{acc,t} = c_i \lambda_{i,i}^t d_i^t + \sum_{j=1,j\neq i}^{n} c_i \cdot \lambda_{j,i}^t d_j^t = c_i \left(\lambda_{i,i}^t d_i^t + \sum_{j=1,j\neq i}^{n} \lambda_{j,i}^t d_j^t \right). \tag{3}$$

3.3. Energy Consumption Models

In this section, we discuss the energy consumption models of UAVs, which include computational energy consumption and communication energy consumption.

3.3.1. Computational Energy Consumption

The computational energy consumption of UAV_i refers to the energy consumption generated by executing all computational tasks on UAV_i.

In general, the computational power consumption of UAV_i (measured in watts) can be expressed as $P_i^{comp} = k \cdot f_i^3$, where k is a constant factor related to the CPU chip architecture, and f_i (measured in cycles/second) is the execution speed of UAV_i [40]. Based on the

previous description, it can be observed that the overall time required to perform all computational tasks on UAV_i during time slot t is

$$T_i^{comp,t} = \frac{C_i^t}{f_i} = \frac{c_i\left(\lambda_{i,i}^t d_i^t + \sum_{j=1,j\neq i}^{n} \lambda_{j,i}^t d_j^t\right)}{f_i}. \tag{4}$$

Then, the computational energy consumption of UAV_i during time slot t can be expressed as

$$E_i^{comp,t} = P_i^{comp} \cdot T_i^{comp,t} = kc_i {f_i}^2\left(\lambda_{i,i}^t d_i^t + \sum_{j=1,j\neq i}^{n} \lambda_{j,i}^t d_j^t\right). \tag{5}$$

3.3.2. Communication Energy Consumption

The communication energy consumption of UAV_i refers to the energy consumption generated by UAV_i during the process of offloading computational tasks to other UAVs.

In a general context, UAVs establish communication with each other using wireless channels within the airspace. According to Shannon's theorem [41], the data transmission rate $R_{i,j}$ (measured in Mbps) for offloading tasks from UAV_i to UAV_j can be calculated as

$$R_{i,j} = B_{i,j}\log_2(1 + \frac{q_{i,j}P_{i,j}^{comm}}{B_{i,j}N_{i,j}}). \tag{6}$$

Here, $B_{i,j}$ represents the channel bandwidth from UAV_i to UAV_j (measured in MHz), $P_{i,j}^{comm}$ stands for the transmission power from UAV_i to UAV_j, $q_{i,j}$ indicates the channel gain from UAV_i to UAV_j (measured in dBm), and $N_{i,j}$ signifies the noise power spectral density from UAV_i to UAV_j (measured in dBm/Hz).

The equation can be rearranged to determine $P_{i,j}^{comm}$ as follows:

$$P_{i,j}^{comm} = \frac{B_{i,j}N_{i,j}(2^{R_{i,j}/B_{i,j}} - 1)}{q_{i,j}}. \tag{7}$$

It can be observed that the transmission time required of UAV_i to offload computational tasks to UAV_j during time slot t is

$$T_{i,j}^{comm,t} = \frac{\lambda_{i,j}^t d_i^t}{R_{i,j} \cdot 10^6}. \tag{8}$$

Hence, the communication energy consumption resulting from offloading computational tasks from UAV_i to UAV_j during time slot t can be expressed as

$$E_{i,j}^{comm,t} = P_{i,j}^{comm} \cdot T_{i,j}^{comm,t} = \frac{\lambda_{i,j}^t d_i^t}{R_{i,j} \cdot 10^6} \cdot \frac{B_{i,j}N_{i,j}\left(2^{R_{i,j}/B_{i,j}} - 1\right)}{q_{i,j}}. \tag{9}$$

Then, we can represent the communication energy consumption of UAV_i during time slot t as

$$E_i^{comm,t} = \sum_{j=1,j\neq i}^{n} E_{i,j}^{comm,t} = \sum_{j=1,j\neq i}^{n}\left(\frac{\lambda_{i,j}^t d_i^t}{R_{i,j} \cdot 10^6} \cdot \frac{B_{i,j}N_{i,j}\left(2^{R_{i,j}/B_{i,j}} - 1\right)}{q_{i,j}}\right). \tag{10}$$

3.3.3. Average Energy Consumption

Based on the models of computational energy consumption and communication energy consumption, the total energy consumption generated by UAV_i during time slot t can be expressed as

$$E_i^t = E_i^{comp,t} + E_i^{comm,t}. \tag{11}$$

Let d^t denote the total size of offloaded computational tasks received by all UAVs during time slot t, i.e., $d^t = \sum_{i=1}^{n} d_i^t$. Next, let $d_i^{t'}$ denote the total size of all computational tasks to be executed by UAV_i at time slot t, i.e.,

$$d_i^{t'} = \lambda_{i,i}^t d_i^t + \sum_{j=1, j\neq i}^{n} \lambda_{j,i}^t d_j^t. \tag{12}$$

Then, we can obtain the average energy consumption at time slot t as

$$E^t = \frac{d_1^{t'}}{d^t} E_1^t + \frac{d_2^{t'}}{d^t} E_2^t + \cdots + \frac{d_n^{t'}}{d^t} E_i^t = \frac{1}{d^t} \sum_{i=1}^{n} d_i^{t'} E_i^t. \tag{13}$$

3.4. Average Response Latency

In this section, we analyze the average response latency of all computational tasks in the computing environment.

In our computing environment, tasks are either executed locally on a UAV or offloaded to other UAVs. This distinction leads to different response latency calculations. Local execution introduces computation latency, while offloading introduces both computation and transmission latency.

Recall that we defined d_i^t as the size of the offload computational tasks received by UAV_i from IOTDs at the beginning of the time slot t. These computational tasks were proportionally distributed by UAV_i, where data of size $\lambda_{i,i}^t d_i^t$ was executed locally on UAV_i, while data of size $\sum_{j=1, j\neq i}^{n} \lambda_{i,j}^t d_i^t$ was offloaded to other UAVs for collaborative execution. It is clear that the response latency of the task assigned to itself by UAV_i in time slot t can be expressed as

$$T_i^{loc,t} = \frac{C_i^{loc,t}}{f_i} = \frac{c_i \lambda_{i,i}^t d_i^t}{f_i}. \tag{14}$$

Since the response latency of the computational tasks that UAV_i offloads to UAV_j for remote execution in time slot t (including computation and transmission latency) can be expressed as

$$T_{i,j}^{\text{off},t} = \frac{c_j \cdot \lambda_{i,j}^t d_i^t}{f_j} + \frac{\lambda_{i,j}^t d_i^t}{R_{i,j} \cdot 10^6}, \tag{15}$$

the average response latency of the computational tasks offloaded by UAV_i for remote execution can be expressed as

$$T_i^{\text{off},t} = \sum_{j=1, j\neq i}^{n} \left(\frac{\lambda_{i,j}^t}{1 - \lambda_{i,i}^t} T_{i,j}^{\text{off},t} \right) = \frac{1}{1 - \lambda_{i,i}^t} \sum_{j=1, j\neq i}^{n} \left(\lambda_{i,j}^t T_{i,j}^{\text{off},t} \right) \tag{16}$$

Then, the average response latency of computational tasks received by UAV_i from IOTDs during time slot t can be expressed as

$$T_i^t = \lambda_{i,i}^t \cdot T_i^{loc,t} + (1 - \lambda_{i,i}^t) T_i^{\mathrm{off},t} = \lambda_{i,i}^t \cdot T_i^{loc,t} + \sum_{j=1, j \neq i}^{n} \lambda_{i,j}^t \cdot T_{i,j}^{\mathrm{off},t}. \tag{17}$$

Therefore, the average response latency of all computational tasks in the computing environment during time slot t can be calculated as

$$T^t = \frac{d_1^t}{d^t} T_1^t + \frac{d_2^t}{d^t} T_2^t + \cdots + \frac{d_n^t}{d^t} T_n^t = \frac{1}{d^t} \sum_{i=1}^{n} d_i^t T_i^t. \tag{18}$$

3.5. Problem Definition

In this section, we formally describe the optimization problem to be solved in this paper.

Given n UAVs, the communication circumstances U_i for all $1 \leq i \leq n$; the execution speed f_i for all $1 \leq i \leq n$; the task related parameters, including d_i^t, c_i for all $1 \leq i \leq n$; the communication-related parameters $R_{i,j}$, $B_{i,j}$, $q_{i,j}$, $N_{i,j}$ for all $1 \leq j \neq i \leq n$; the constant factor related to the CPU chip architecture k; and the latency constraint T^*, obtain the optimal offloading scheme $\left(\lambda_{i,1}^t, \lambda_{i,2}^t, \ldots, \lambda_{i,n}^t\right)$ for each UAV in every time slot, aiming to minimize average energy consumption while ensuring compliance with the average response latency constraint. The optimization problem can be formulated as follows:

$$\begin{aligned} &\mathrm{P}_1 : \min E^t, \\ &\text{s.t. } C1 : \begin{cases} \lambda_{i,j}^t = 0, & \text{if } \mathrm{UAV}_i^j = 0; \\ \lambda_{i,j}^t \in [0,1], & \text{if } \mathrm{UAV}_i^j = 1. \end{cases} \\ &C2 : \lambda_{i,1}^t + \lambda_{i,2}^t + \cdots + \lambda_{i,n}^t = 1, \text{for all } 1 \leq i \leq n, \\ &C3 : T^t \leq T^*, \text{for all } t \in \{1, 2, 3, \ldots, T\}. \end{aligned} \tag{19}$$

In this formulation, constraint $C1$ denotes the proportion of tasks that each UAV offloads to other UAVs within its communication range as a value in the range [0, 1]. Constraint $C2$ ensures that each UAV's computational task has an appropriate executor and that all tasks are executed. Constraint $C3$ specifies that the average response latency of the UAVs in completing tasks within each time slot must be less than the defined latency constraint, represented as T^*.

4. Our Solutions

4.1. Problem Transformation

In a dynamic UAV-assisted MEC environment, system state transitions at each time slot t are contingent upon the real-time system state and the current task offloading policy. Consequently, the offloading process within the UAV-assisted MEC environment constitutes an MAMDP [42]. To address this optimization problem, we naturally consider the application of deep reinforcement learning to train intelligent agents. These agents can be deployed on edge service controllers to generate task offloading strategies based on the observed system state. Subsequently, UAVs execute these task offloading actions, and rewards are allocated to the agent upon successful execution. Meanwhile, the system proceeds to the next state, and the agent refines its policy by leveraging the experiences gained from these states, actions, and rewards. In this paper, we define the terms state, observation, action, and reward as follows.

State: The environment state is the information used to determine subsequent actions, observations, and rewards, and it is a function of historical data. In the context of UAV-assisted MEC environment, we define the environmental state for time slot t as follows:

$$s^t = \{d_1^t, d_2^t, \ldots, d_n^t, G_1, G_2, \ldots G_n\}. \tag{20}$$

We denote the set G_i to encompass various attributes of UAV_i and communication-related information as follows:

$$G_i = \{f_i, c_i, k, B_{i,1}, \ldots, B_{i,n}, R_{i,1}, \ldots, R_{i,n}, q_{i,1}, \ldots q_{i,n}, N_{i,1}, \ldots, N_{i,n}, U_i\}. \tag{21}$$

Observation: In our framework, we consider the data size of tasks received by UAVs as our observation. The observation for time slot t can be expressed as

$$O^t = \{d_1^t, d_2^t, \ldots, d_n^t\}. \tag{22}$$

Action: In the context of a UAV-assisted MEC environment, agent interactions are facilitated through a predefined set of actions that encompass all possible choices. At time slot t, each agent's action involves determining the proportions $\lambda_{i,j}^t$ by which UAV_i offloads tasks to UAV_j within its communication range. Consequently, the action of UAV_i at time slot t is represented as

$$a_i^t = \left(\lambda_{i,1}^t, \lambda_{i,2}^t, \ldots, \lambda_{i,n}^t\right). \tag{23}$$

Given the presence of n agents in the UAV-assisted MEC environment, we can derive the set of actions for all n agents at time slot t as follows

$$a^t = \{a_1^t, a_2^t, \ldots a_n^t\}. \tag{24}$$

Reward: The reward is a function that assesses the impact of an action taken by an entity in a specific state. In the context of the UAV-assisted MEC environment, our objective is to ensure that the agent's task offloading policy complies with the latency constraint T^* while minimizing the average energy consumption. To achieve this, we define the following reward function

$$r^t = \begin{cases} \frac{\omega_e}{E^t}, & \text{if } T^t \leq T^*; \\ \omega_t\left(T^* - T^t\right), & \text{if } T^t > T^*. \end{cases} \tag{25}$$

In this context, ω_e and ω_t represent the coefficients of the reward function. Given that agents in the UAV-assisted MEC environment collaborate to achieve optimization objectives, this reward function is shared among all agents in this environment.

4.2. Preliminaries of the MADDPG

Reinforcement learning aims to find the optimal decision in uncertain environments on the basis of qualitative and noisy on-line performance feedback provided by the environments [43]. The agent corrects its policy in the environment by continuously interacting iteratively, allowing the agent to learn the best or near-optimal solution by maximizing the expected cumulative reward [44,45].

We employ the multi-agent deep reinforcement learning algorithm MADDPG to collaboratively optimize the objective function defined in Equation (19). As depicted in Figure 2, this algorithm integrates the deep deterministic policy gradient (DDPG) within a cooperative multi-agent learning framework, where each agent receives the same immediate reward. The MADDPG is built upon the actor–critic framework, leveraging both centralized training and decentralized execution.

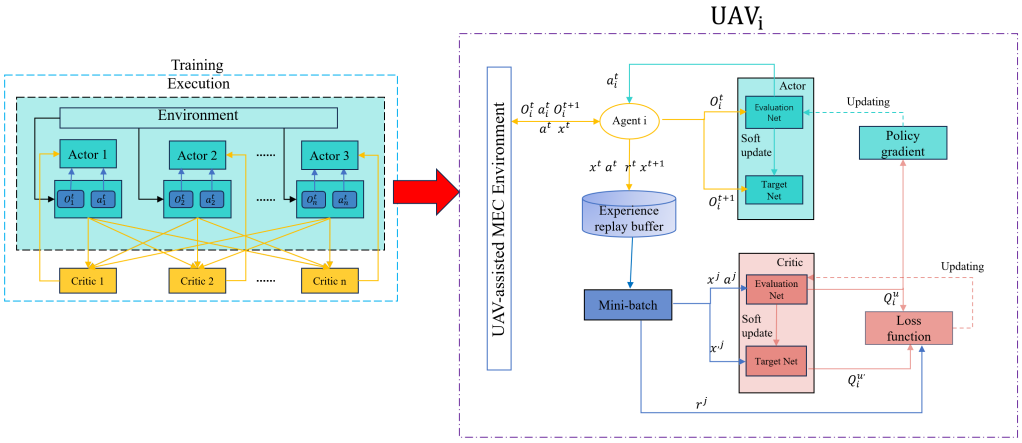


Figure 2. The MADDPG framework in UAV-assisted MEC environment.

The MADDPG algorithm introduces several enhancements to both the actor–critic and DDPG algorithms, addressing the limitations of traditional reinforcement learning in coping with complex multi-agent environments while preserving DDPG's efficiency and applicability to continuous action spaces. In traditional reinforcement learning, each agent is continually learning to enhance its policy, resulting in a dynamically unstable environment from the perspective of each agent. This dynamic instability makes the network environment non-stationary, thereby violating the markov assumptions necessary for stable and convergent learning.

Unlike traditional single-agent algorithms, the MADDPG involves n agents. Let $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ represent the set of strategies adopted by all agents for selecting their actions based on their observations, and let $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ denote the parameter set of networks. Each agent updates its policy independently with the aim of maximizing the expected cumulative reward. The expected cumulative reward of agent i is as follows:

$$J(\theta_i) = E_{s\sim p^{\pi}, a_i\sim\pi_i}\left[\sum_{t=1}^{T}\gamma^{t-1}r^t\right]. \tag{26}$$

In this context, r^t represents the reward obtained by agent i at time slot t. p^{π} signifies the state distribution under the policy π. γ stands for the discount factor, utilized to discount future rewards. The strategy gradient, a parameter that directly adjusts the strategy gradient, aims to maximize $J(\theta_i)$ by taking steps in the direction of $\nabla_{\theta_i}J(\theta_i)$. To maintain clarity, we will exclude the time index in the forthcoming equations. In the context of a stochastic strategy, the resulting strategy gradient formula is as follows:

$$\nabla_{\theta_i}J(\theta_i) = E_{s\sim\rho^{\pi}, a_i\sim\pi_i}\left[\nabla_{\theta_i}\log\pi_i(a_i \mid o_i)Q_i^{\pi}(x, a_1, a_2\cdots, a_n)\right]. \tag{27}$$

Here, $Q_i^{\pi}(x, a_1, \cdots, a_n)$ is a centralized action–value function that takes as inputs the actions of all agents, $a_1, a_2\cdots, a_n$ in addition to some state information x, and outputs the Q-value for agent i. x represents the observations made by all agents, referred to as $x = [o_1, o_2, \ldots, o_n]$.

In contrast to random strategies, deterministic strategies dictate specific actions in particular states. Extending the preceding stochastic strategy to a deterministic strategy, denoted as μ_{θ_i} (abbreviated as μ_i), allows us to derive the gradient of the deterministic strategy as follows:

$$\nabla_{\theta_i}J(\mu_i) = E_{x,a\sim D}\left[\nabla_{\theta_i}\mu_i(o_i)\nabla_{a_i}Q_i^{\mu}(x, a_1, a_2, \cdots, a_n)\Big|_{a_i=\mu_i(o_i)}\right]. \tag{28}$$

D is the experience replay buffer, retaining all the data generated through the agent's interactions with the environment, each composed of $\left(x, x^{'}, a_1, a_2, \ldots, a_n, r\right)$, where $x^{'}$ is all agent observations at the next moment.

During the training phase, the parameters of both the actor and critic networks are updated by randomly sampling a mini-batch of size κ from the experience replay buffer. The gradient for the actor network of agent i can be computed using the following equation:

$$\nabla_{\theta_i} J(\mu_i) \approx \frac{1}{\kappa} \sum_j \nabla_{\theta_i} \mu_i \left(o_i^j\right) \nabla_{a_i} Q_i^{\mu}(x^j, a_1^j, a_2^j, \ldots, a_n^j)\Big|_{a_i=\mu_i\left(o_i^j\right)}, \tag{29}$$

where j is the index of the sample. The critic network of agent i can be updated by minimizing the loss function as follows:

$$L(\theta_i) \approx \frac{1}{\kappa} \sum_j \left(y^j - Q_i^{\mu}\left(x^j, a_1^j, a_2^j \ldots, a_n^j\right)\right)^2, \tag{30}$$

where y^j is the sum of the agent's immediate reward and discount target Q-value. y^j is obtained from the following equation:

$$y^j = r^j + \gamma Q_i^{\mu^{'}}\left(x^{'j}, a_1^{'j}, a_2^{'j}, \cdots, a_n^{'j}\right)\Big|_{a_k^{'}=\mu_k^{'}\left(o_k^{'j}\right)}, \tag{31}$$

where $\mu^{'} = \left\{\mu_{\theta_1^{'}}, \mu_{\theta_2^{'}}, \ldots, \mu_{\theta_n^{'}}\right\}$ is the set of target policies with delayed parameters $\theta_i^{'}$. To enhance the stability of the network training process, we will implement a soft update strategy in which the target network parameters $\theta_i^{'}$ will be updated based on the current network parameters θ_i according to the following equation:

$$\theta_i^{'} \leftarrow (1-\varepsilon)\theta_i^{'} + \varepsilon\theta_i, \tag{32}$$

where $0 < \varepsilon \ll 1$, which represents the smoothing factor. Upon completion of training, each agent can independently choose its actions based on local observations without knowledge of other agents' information.

Algorithm 1 summarizes the detailed flow of the MADDPG algorithm for UAV-assisted MEC environment proposed in this paper. First, we randomly initialize the agent's networks (lines 1–2) and initialize our replay buffer memory (line 3). The agents in the environment obtain the initial state of the observation, and the agent selects the task offloading percentage of UAVs within the communication range according to its policy and exploration noise (line 8). We employ Gaussian noise to enhance the action detection strategy, where ψ represents the Gaussian noise. After the agent performs the selected action, the agent obtains a step reward, the next observation (lines 9–10). Then, the transition experiences of each agent are stored in the replay buffer memory, and the scenario state is updated to the next state (lines 11–12). The critic and actor network parameters of each agent are updated based on Equations (30) and (29), respectively (lines 16–17). Then, the target network parameters for each agent are updated according to Equation (32). The algorithm is terminated when it reaches the defined maximum number of sets.

Algorithm 1 MADDPG-Based Task Offloading Optimization in UAV-assisted MEC Environment

Input:
$\{\mathrm{UAV}_1, \mathrm{UAV}_2, \ldots, \mathrm{UAV}_n\}$,$U = \{U_1, U_2, \ldots, U_n\}$
Output:
$\left[(\lambda^t_{1,1}, \lambda^t_{1,2}, \ldots, \lambda^t_{1,n}), \ldots, (\lambda^t_{n,1}, \lambda^t_{n,2}, \ldots, \lambda^t_{n,n})\right]$

Randomly initialize the weights of actor and critic networks
Randomly initialize the weights of target actor and critic networks
Initialize the size of replay buffer memory
for episode $e = 1 : EP$ **do**
  Initialize a random process ψ for action exploration
  Receive initial observations
  for step $t = 1 : ST$ **do**
    For each agent i, select action $a^t_i = \mu_i(O^t_i) + \psi^t$ w.r.t the exploration and current policy
    Execute the actions $a^t = \{a^t_1, a^t_2, \ldots a^t_n\}$
    Obtain global reward r^t, new observations O^{t+1}_i
    Store the experience transitions in the replay buffer memory
    $x \leftarrow x^{'}$
    for agent $i = 1$ to n **do**
      Randomly select a mini-batch of experience transitions κ from the replay buffer memory
      Set y^j according to Equation (31)
      Update the critic network through minimizing the loss based on Equation (30)
      Update actor network utilizing the sampled policy gradient based on Equation (29)
    End for
    Update target network parameters via Equation (32)
  End for
End for

5. Performance Evaluation

We performed a series of simulations to assess the performance of the MADDPG-based task offloading algorithm. Initially, we examined the impact of various parameters on the MADDPG scheme's performance. Subsequently, we compared the MADDPG-based task offloading algorithm with four other schemes in diverse environments.

5.1. Simulation Settings

To assess the performance of our algorithm, we conducted simulations in two distinct scenarios involving six UAVs and eight UAVs, respectively. In these scenarios, each UAV follows a predefined flight path during independent operations. Positioned at fixed coordinates, the UAVs offer computational offloading services to IOTDs within their coverage zones. These UAVs receive computational tasks from IOTDs, with task sizes ranging from 10 Mb to 25 Mb. The parameters used in our experiments are as follows: constant factor related to the CPU chip architecture $k = 10^{-27}$, number of CPU cycles needed to execute 1-bit data $c_i \in \{400, 500, 600, 800\}$ cycles/bit, execution speed $f_i = 2.0 + 0.4(i-1)$ GHz, channel bandwidth $B_{i,j} = 3.0 + 0.1(j-1)$ MHz, data transmission rate $R_{i,j} = 10 + 0.5(j-1)$ Mbps, noise power spectral density $N_{i,j} = -174 - 0.2(j-1)$ dBm/Hz; all of the above i and j take a range of values $1 \leq j \neq i \leq n$, where n is the number of UAVs in the current environment.

Our simulations were implemented in the PyCharm development environment using PyTorch and Python 3.7.3. In our algorithm, each agent is equipped with both a critic network and a actor network. these networks are four-layer fully connected neural networks, commonly referred to as multilayer perceptrons. Specifically, the hidden layers in these fully connected neural networks were set to $[512, 256]$ and $[256, 128]$, respectively, with ReLU

serving as the activation function in the hidden layers. The important hyperparameters used in the experiments are shown in Table 1.

Table 1. Experimental parameters.

Parameter	Definition	Setting
EP	Episode	1,000,000
D	Replay buffer size	1,000,000
ω_e	Coefficient for reward function	1000
ω_t	Coefficient for reward function	10
σ	Gaussian noise variance	0.05∼0.5
ST	Steps	1
γ	Discount factor	0.1
α	Critic network learning rate	0.01
β	Actor network learning rate	0.01
ε	Softupdate factor	0.01
p_i	Number of iterations	200
p_n	Number of particles	100
c_1	Acceleration coefficients	1.5
c_2	Acceleration coefficients	1.5
p_w	Inertia weight	0.5

5.2. Parameter Analysis

The training regimen for our scheme unfolds as follows: The initiation process encompasses the initialization of the agent network and the replay buffer memory. The agents within the environment receive initial observations. Each agent independently selects the task offloading ratios for UAVs within their communication range, guided by a combination of their individual strategies and exploration noise. Following the execution of their chosen actions, agents receive step rewards and the subsequent set of observations. The experiences of each agent are recorded within the replay buffer memory. Concurrently, the environment's state transitions to the subsequent state. Agents update their network parameters based on their interactions with the environment. The algorithm will terminate when the predefined maximum number of episodes is reached.

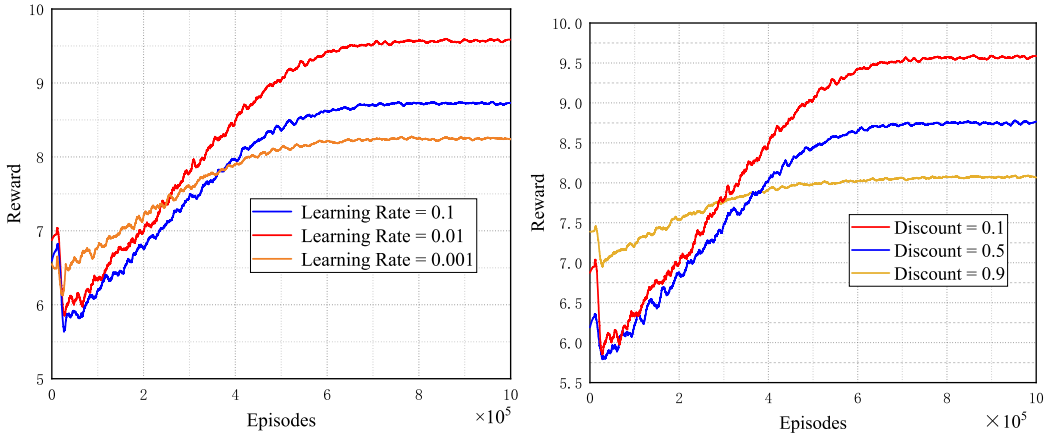
During the training process, we investigate the impact of learning rates and discount factors on the convergence of the MADDPG-based scheme proposed in this study.

5.2.1. Impact of Learning Rate on Convergence

Figure 3a depicts the convergence performance of the Adam optimizer in the MADDPG algorithm for three different learning rates. As seen in Figure 3a, the total reward for all learning rates increases with increasing learning time and converges within a specific range. This is because, as the learning episodes increase, the actor and critic networks adjust their parameters to fit the optimal strategy and action values. Furthermore, we can find that setting the learning rate to 0.01 obtains rewards higher than 0.1 and 0.001. This is because the learning rate determines the step size of the change in the weights with the direction of the gradient. The larger the learning rate, the more significant the change in weights. Therefore, a large or small learning rate can make the network fall into a local optimum solution.

5.2.2. Impact of Discount Factor on Convergence

Figure 3b depicts the convergence performance of three different discount factors in the MADDPG algorithm. In this experiment, the discount factor γ is set to 0.1, 0.5, and 0.9, respectively. It can be seen from the figure that the algorithm converges with the highest reward when the discount factor is set to 0.1. This is because the discount factor impacts the balance between current and future rewards. Given our optimization objective, which prioritizes immediate rewards, setting the discount factor to 0.1 results in higher rewards.



(**a**) (**b**)

Figure 3. Convergence performances of MADDPG with different parameters. (**a**) Different learning rates. (**b**) Different discount factors.

5.3. Comparison Experiments

To evaluate the efficacy of our solution in the context of the UAV-assisted MEC environment, we conduct comparisons with the following benchmark solutions in scenarios involving six and eight UAVs, respectively.

- PSO scheme: We treat the set of offloading decisions made by all UAVs in the current environment as a particle, utilizing Equation (25) as the fitness function for the PSO algorithm. The parameters for the PSO algorithm are presented in Table 1.
- URM scheme: In the URM scheme, each UAV first determines the number of UAVs within its communication range (denoted as m). It then distributes the tasks equally between itself and the other UAVs within its communication range. Thus, the size of the task that UAV$_i$ shares with other UAVs in its communication range is $\frac{d_i^t}{m+1}$.
- Proximal policy optimization (PPO) scheme: Each UAV has an independent PPO [46] model and each UAV makes independent offloading decisions.
- DDPG scheme: Each UAV operates autonomously with its independent DDPG model, devoid of a centralized critic network and local information sharing. Each UAV autonomously makes offloading decisions. We employ the neural network structure consistent with the MADDPG algorithm for both the actor and critic networks of the DDPG.

We employ the MADDPG algorithm alongside PPO and DDPG algorithms to iteratively interact with the environment and update network parameters within a specified environment. This allows us to conduct a comparative analysis of their convergence performance by recording the average reward per 100 episodes.

As depicted in Figure 4, we have compared the convergence performance of our approach with that of the PPO and DDPG algorithms across various environments. In the initial phase, the average reward is relatively low due to the random initialization of network parameters. During the initial 20,000 training episodes, both the MADDPG and DDPG algorithms interacted with the environment to accumulate experiences for each agent, which were subsequently stored in the replay buffer memory. After completing these initial 20,000 episodes, network parameters are updated based on the data stored in the replay buffer memory. It is essential to note that the PPO algorithm operates as an online algorithm, necessitating the generation of data using the current policy and updating the policy based on this data. As depicted in Figure 4, MADDPG outperforms DDPG and PPO in terms of average reward. MADDPG achieves these outstanding results because during training, the critic of each agent in MADDPG has access to information regarding the actions of other agents.

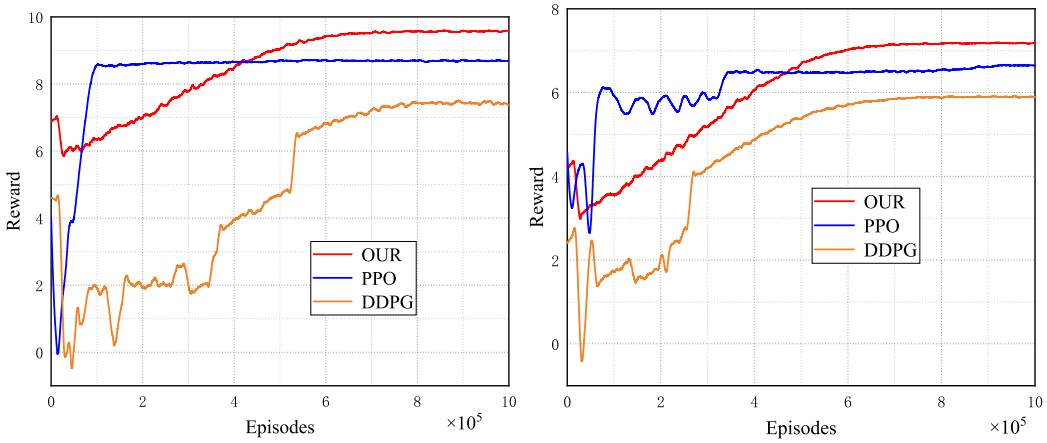


Figure 4. Convergence performance of our approach, DDPG, and PPO in various environments. (**a**) Six-UAV environment. (**b**) Eight-UAV environment.

Based on the environmental parameters in Section 5.1, we randomly initialized five states to assess the performance gap between our scheme and other benchmark algorithms. These states consist of sets of task sizes received by UAVs from the IOTDs. These tasks vary in size from 10 Mb to 25 Mb and are generated following a uniform distribution. Within each state, the offloading policies prescribed by our scheme and the other benchmark algorithms are evaluated, and the average energy consumption and average task execution latency for various offloading policies are computed.

Figure 5 illustrates the average latency and average energy consumption of offloading policies provided by five different algorithms in environments with six and eight UAVs across five distinct states. The latency constraints in the environments with six and eight UAVs were 1.45 s and 0.95 s, respectively. Figure 5a,b show the simulation environment with six UAVs, and Figure 5c,d show the simulation environment with eight UAVs. We observe a significant reduction in the average response latency for performing computational tasks received from IOTDs within each time slot as we increase the computational resources, specifically the number of UAVs. It is also relatively easy to find that the average energy consumption required for the computation is much higher because the increase in the number of UAVs means more computational tasks from IOTDs are accepted. From Figure 5a,c, we can easily find that the offloading decisions made by the MADDPG algorithm satisfy the latency constraint for the random five states. Figure 5b,d demonstrate that the average energy consumption for processing the computational tasks using the MADDPG algorithm consistently outperforms the other baseline algorithms.

Figure 5 shows the comparative effect of our algorithm and the baseline algorithm for only five random states. To evaluate the real effectiveness of the algorithms in practical environments, we expanded the initial five random states to a total of 1000 random states. Employing the same methodology as described above, we collected offloading strategies from various schemes across different states. Subsequently, we computed the average energy consumption and average latency associated with these offloading strategies for task execution. It is worth noting that we only calculate the sum of the average energy consumption when all algorithms meet the latency constraints, as there may be cases of violation.

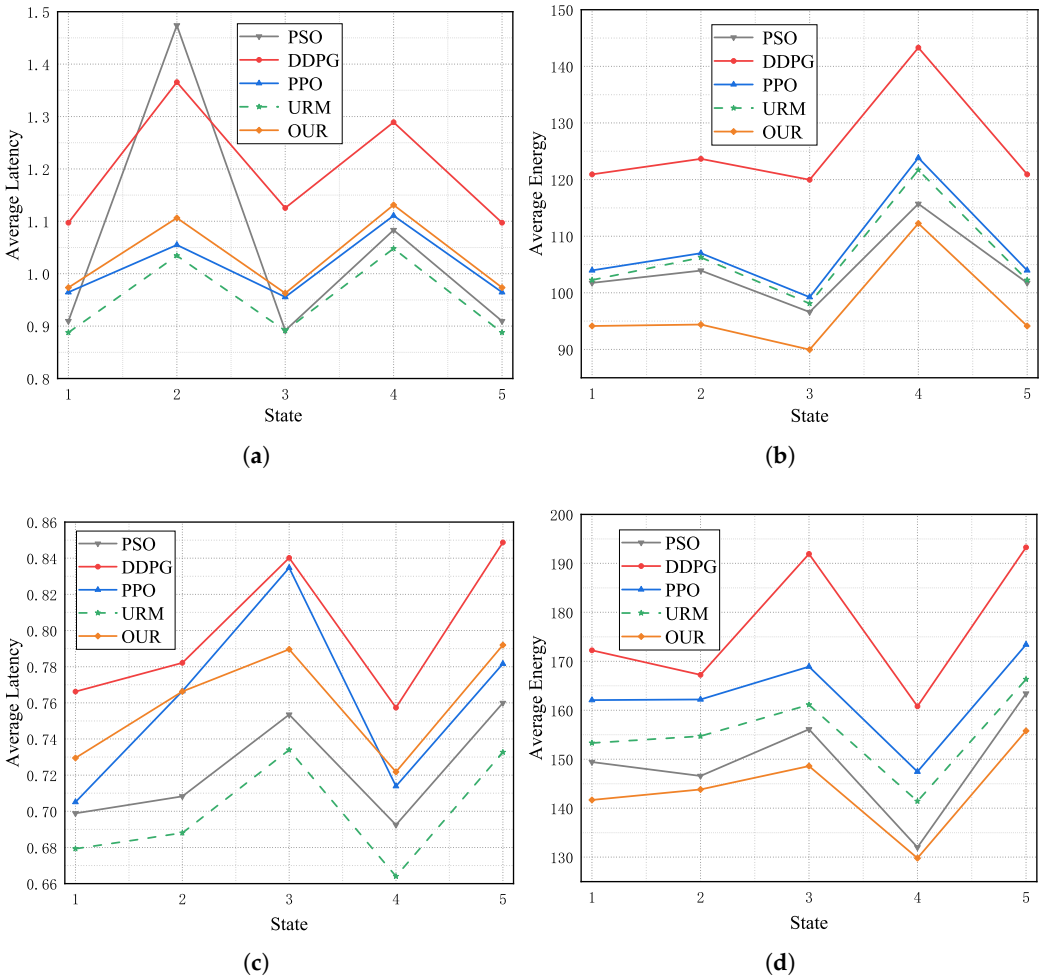


Figure 5. Average latency and average energy consumption of different schemes. (**a**) Six-UAV environment. (**b**) Six-UAV environment. (**c**) Eight-UAV environment. (**d**) Eight-UAV environment.

In environments featuring six and eight UAVs, the DDPG baseline algorithm breached latency constraints 187 and 3 times, respectively. The PSO baseline algorithm exceeded these constraints 233 and 90 times in the corresponding scenarios, while the PPO baseline algorithm did so 3 and 16 times in the same contexts. Notably, neither the MADDPG algorithm nor the URM algorithm violated the latency constraint. Figure 6 displays the total average energy consumption of the five algorithms across 1000 random states, where all algorithms successfully adhere to the latency constraints. The results indicate that our scheme, employing the MADDPG algorithm, achieves lower average energy consumption in comparison to the benchmark algorithm. For instance, with $n = 6$, the offloading scheme offered by MADDPG exhibits an average energy consumption that is 21.06% lower than that of DDPG, 16.72% lower than that of PPO, 8.93% lower than that of URM, and 6.99% lower than that of PSO.

In conclusion, our scheme, employing the MADDPG algorithm, effectively reduces average energy consumption while adhering to latency constraints in diverse environments. This affirms the efficacy of our approach across various scenarios.

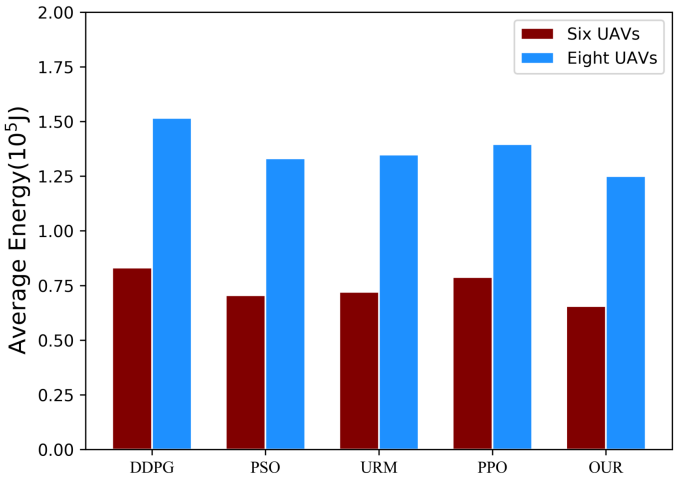


Figure 6. The total average energy consumption of various schemes meeting the latency constraint in 1000 random states.

6. Conclusions

In this paper, we investigate the problem of collaborative computation offloading involving multiple UAVs under the constraints of limited communication range and latency. Our primary objectiveis is to minimize the average energy consumption of these UAVs while ensuring latency constraints are met. To address the high dynamism and complexity of this problem, we formulate the cooperative offloading problem as an MAMDP and use a cooperative algorithm based on the MADDPG to tackle it effectively. The MADDPG algorithm significantly reduces energy consumption in UAV-assisted MEC environment by facilitating cooperative computational offloading while meeting the latency constraints. Simulation results demonstrate the superiority of the MADDPG algorithm over existing methods. It is worth noting that in this article we simplified the process through which UAVs receive tasks from IOTDs. In our upcoming research, we will delve into the optimization of computational offloading for IOTDs.

Author Contributions: Conceptualization, Z.X. and J.L.; Methodology, Z.X.; Software, Z.X., J.L. and Y.D.; Validation, Y.G. and Y.D.; Formal analysis, J.L. and Z.H.; Investigation, Z.X., J.L. and Z.H.; Resources, Z.H.; Data curation, Y.G. and Y.D.; Writing—original draft, Z.X. and J.L.; Writing—review and editing, J.L. and Z.H.; Visualization, Z.X.; Supervision, Z.H.; Project administration, Z.H.; Funding acquisition, Z.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Applied Basic Research Foundation of Yunnan Province under Grant Nos. 202201AT070156 and 202301AT070194, in part by the Open Foundation of Yunnan Key Laboratory of Software Engineering under Grant No 2023SE208, and in part by the Graduate Student Research Innovation Project of Yunnan University under Grant No. ZC-22221533.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Mathematical Notations and Definitions.

Symbol	Definition
n	Number of UAVs
U_i	Communication circumstances of UAV$_i$
d_i^t	Data size of tasks received by UAV$_i$ from IOTDs at time slot t

$\lambda_{i,i}^{t}$	The percentage of tasks that UAV$_i$ handles locally at time slot t
$\lambda_{i,j}^{t}$	The percentage of tasks that UAV$_i$ offload to UAV$_j$ at time slot t
c_i	Number of CPU cycles needed to execute 1-bit data on UAV$_i$
$C_i^{loc,t}$	CPU cycles required by UAV$_i$ to perform local tasks at time slot t
$C_i^{acc,t}$	CPU cycles required by UAV$_i$ to perform tasks from other UAVs at time slot t
C_i^{t}	Total CPU cycles required by UAV$_i$ to perform all tasks at time slot t
P_i^{comp}	Computational power consumption of UAV$_i$
k	Constant factor related to the CPU chip architecture
f_i	Execution speed of UAV$_i$
$T_i^{comp,t}$	Overall time required to perform all tasks on UAV$_i$ during time slot t
$E_i^{comp,t}$	Energy consumption required to perform all tasks on UAV$_i$ during time slot t
$R_{i,j}$	Data transmission rate between UAV$_i$ and UAV$_j$
$B_{i,j}$	Channel bandwidth from UAV$_i$ to UAV$_j$
$P_{i,j}^{comm}$	Transmission power from UAV$_i$ to UAV$_j$
$q_{i,j}$	Channel gain from UAV$_i$ to UAV$_j$
$N_{i,j}$	Noise power spectral density from UAV$_i$ to UAV$_j$
$T_{i,j}^{comm,t}$	Transmission time required of UAV$_i$ to offload tasks to UAV$_j$ during time slot t
$E_{i,j}^{comm,t}$	Energy consumption due to task offloading from UAV$_i$ to UAV$_j$ during time slot t
$E_i^{comm,t}$	Communication energy consumption of UAV$_i$ during time slot t
E_i^{t}	Total energy consumption of the UAV$_i$ during time slot t
d^t	Data size of the tasks on all UAVs at time slot t
$d_i^{t'}$	Data size of all tasks to be executed by UAV$_i$ at time slot t
E^t	Average energy consumption at time slot t
$T_i^{loc,t}$	Response latency of the task assigned to itself by UAV$_i$ in time slot t
$T_{i,j}^{\mathrm{off},t}$	Response latency of tasks that UAV$_i$ offloads to UAV$_j$ for execution in time slot t
$T_i^{\mathrm{off},t}$	Average response latency of tasks offloaded by UAV$_i$ for execution in time slot t
T_i^{t}	Average response latency of tasks received by UAV$_i$ from IOTDs in time slot t
T^t	Average response latency of all tasks during time slot t
T^*	Latency constraint

References

1. Zhao, M.; Li, W.; Bao, L.; Luo, J.; He, Z.; Liu, D. Fairness-Aware Task Scheduling and Resource Allocation in UAV-Enabled Mobile Edge Computing Networks. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 2174–2187. [CrossRef]
2. Mach, P.; Becvar, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
3. Dao, N.N.; Pham, Q.V.; Tu, N.H.; Thanh, T.T.; Bao, V.N.Q.; Lakew, D.S.; Cho, S. Survey on Aerial Radio Access Networks: Toward a Comprehensive 6G Access Infrastructure. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1193–1225. [CrossRef]
4. Lakew, D.S.; Sa'ad, U.; Dao, N.-N.; Na, W.; Cho, S. Routing in Flying Ad Hoc Networks: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1071–1120. [CrossRef]
5. Zhou, F.; Hu, R.Q.; Li, Z.; Wang, Y. Mobile Edge Computing in Unmanned Aerial Vehicle Networks. *IEEE Wirel. Commun.* **2020**, *27*, 140–146. [CrossRef]
6. Qiu, J.; Grace, D.; Ding, G.; Zakaria, M.D.; Wu, Q. Air-Ground Heterogeneous Networks for 5G and Beyond via Integrating High and Low Altitude Platforms. *IEEE Wirel. Commun.* **2019**, *26*, 140–148. [CrossRef]
7. Kurt, G.K.; Khoshkholgh, M.G.; Alfattani, S.; Ibrahim, A.; Darwish, T.S.; Alam, M.S.; Yanikomeroglu, H.; Yongacoglu, A. A Vision and Framework for the High Altitude Platform Station (HAPS) Networks of the Future. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 729–779. [CrossRef]
8. Motlagh, N.H.; Bagaa, M.; Taleb, T. Energy and Delay Aware Task Assignment Mechanism for UAV-Based IoT Platform. *IEEE Internet Things J.* **2019**, *6*, 6523–6536. [CrossRef]
9. Jeong, S.; Simeone, O.; Kang, J. Mobile Edge Computing via a UAV-Mounted Cloudlet: Optimization of Bit Allocation and Path Planning. *IEEE Trans. Veh. Technol.* **2018**, *67*, 2049–2063. [CrossRef]
10. Zhang, T.; Xu, Y.; Loo, J.; Yang, D.; Xiao, L. Joint Computation and Communication Design for UAV-Assisted Mobile Edge Computing in IoT. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5505–5516. [CrossRef]
11. Diao, X.; Guan, X.; Cai, Y. Joint Offloading and Trajectory Optimization for Complex Status Updates in UAV-Assisted Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 23881–23896. [CrossRef]
12. Lv, W.; Yang, P.; Zheng, T.; Yi, B.; Ding, Y.; Wang, Q.; Deng, M. Energy Consumption and QoS-Aware Co-Offloading for Vehicular Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 5214–5225. [CrossRef]

13. Chen, J.; Wu, H.; Li, R.; Jiao, P. Green Parallel Online Offloading for DSCI-Type Tasks in IoT-Edge Systems. *IEEE Trans. Ind. Inform.* **2022**, *18*, 7955–7966. [CrossRef]
14. Hu, X.; Masouros, C.; Wong, K.-K. Reconfigurable Intelligent Surface Aided Mobile Edge Computing: From Optimization-Based to Location-Only Learning-Based Solutions. *IEEE Trans. Commun.* **2021**, *69*, 3709–3725. [CrossRef]
15. Guo, M.; Wang, W.; Huang, X.; Chen, Y.; Zhang, L.; Chen, L. Lyapunov-Based Partial Computation Offloading for Multiple Mobile Devices Enabled by Harvested Energy in MEC. *IEEE Internet Things J.* **2022**, *9*, 9025–9035. [CrossRef]
16. Luo, Q.; Li, C.; Luan, T.H.; Shi, W. Minimizing the Delay and Cost of Computation Offloading for Vehicular Edge Computing. *IEEE Trans. Serv. Comput.* **2022**, *15*, 2897–2909. [CrossRef]
17. Abbas, N.; Fawaz, W.; Sharafeddine, S.; Mourad, A.; Abou-Rjeily, C. SVM-Based Task Admission Control and Computation Offloading Using Lyapunov Optimization in Heterogeneous MEC Network. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 3121–3135. [CrossRef]
18. Sheng, M.; Dai, Y.; Liu, J.; Cheng, N.; Shen, X.; Yang, Q. Delay-Aware Computation Offloading in NOMA MEC Under Differentiated Uploading Delay. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 2813–2826. [CrossRef]
19. Moura, J.; Hutchison, D. Game Theory for Multi-Access Edge Computing: Survey, Use Cases, and Future Trends. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 260–288. [CrossRef]
20. Zhou, J.; Tian, D.; Sheng, Z.; Duan, X.; Shen, X. Distributed Task Offloading Optimization with Queueing Dynamics in Multiagent Mobile-Edge Computing Networks. *IEEE Internet Things J.* **2021**, *8*, 12311–12328. [CrossRef]
21. Pham, X.-Q.; Huynh-The, T.; Huh, E.-N.; Kim, D.-S. Partial Computation Offloading in Parked Vehicle-Assisted Multi-Access Edge Computing: A Game-Theoretic Approach. *IEEE Trans. Veh. Technol.* **2022**, 71, 10220–10225. [CrossRef]
22. Xiao, Z.; Dai, X.; Jiang, H.; Wang, D.; Chen, H.; Yang, L.; Zeng, F. Vehicular Task Offloading via Heat-Aware MEC Cooperation Using Game-Theoretic Method. *IEEE Internet Things J.* **2020**, *7*, 2038–2052. [CrossRef]
23. Chen, Y.; Zhao, J.; Wu, Y.; Huang, J.; Shen, X.S. QoE-Aware Decentralized Task Offloading and Resource Allocation for End-Edge-Cloud Systems: A Game-Theoretical Approach. *IEEE Trans. Mob. Comput.* **2022**. [CrossRef]
24. Teng, H.; Li, Z.; Cao, K.; Long, S.; Guo, S.; Liu, A. Game Theoretical Task Offloading for Profit Maximization in Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, *22*, 5313–5329. [CrossRef]
25. Fang, T.; Wu, D.; Chen, J.; Liu, D. Cooperative Task Offloading and Content Delivery for Heterogeneous Demands: A Matching Game-Theoretic Approach. *IEEE Trans. Cogn. Commun. Netw.* **2022**, *8*, 1092–1103. [CrossRef]
26. You, F.; Ni, W.; Li, J.; Jamalipour, A. New Three-Tier Game-Theoretic Approach for Computation Offloading in Multi-Access Edge Computing. *IEEE Trans. Veh. Technol.* **2022**, *71*, 9817–9829. [CrossRef]
27. Zhang, K.; Gui, X.; Ren, D.; Li, D. Energy–Latency Tradeoff for Computation Offloading in UAV-Assisted Multiaccess Edge Computing System. *IEEE Internet Things J.* **2021**, *8*, 6709–6719. [CrossRef]
28. Hussain, F.; Hassan, S.A.; Hussain, R.; Hossain, E. Machine Learning for Resource Management in Cellular and IoT Networks: Potentials, Current Solutions, and Open Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1251–1275. [CrossRef]
29. Wang, J.; Hu, J.; Min, G.; Zomaya, A.Y.; Georgalas, N. Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 242–253. [CrossRef]
30. Liu, Y.; Yu, H.; Xie, S.; Zhang, Y. Deep Reinforcement Learning for Offloading and Resource Allocation in Vehicle Edge Computing and Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 11158–11168. [CrossRef]
31. Zhou, H.; Jiang, K.; Liu, X.; Li, X.; Leung, V.C.M. Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 1517–1530. [CrossRef]
32. Tang, M.; Wong, V.W.S. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Trans. Mob. Comput.* **2022**, *21*, 1985–1997. [CrossRef]
33. Dai, Y.; Zhang, K.; Maharjan, S.; Zhang, Y. Deep Reinforcement Learning for Stochastic Computation Offloading in Digital Twin Networks. *IEEE Trans. Ind. Inform.* **2021**, *17*, 4968–4977. [CrossRef]
34. Gao, H.; Huang, W.; Liu, T.; Yin, Y.; Li, Y. PPO2: Location Privacy-Oriented Task Offloading to Edge Computing Using Reinforcement Learning for Intelligent Autonomous Transport Systems. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 7599–7612. [CrossRef]
35. Wang, J.; Hu, J.; Min, G.; Zhan, W.; Ni, Q.; Georgalas, N. Computation Offloading in Multi-Access Edge Computing Using a Deep Sequential Model Based on Reinforcement Learning. *IEEE Commun. Mag.* **2019**, *57*, 64–69. [CrossRef]
36. Lu, H.; He, X.; Du, M.; Ruan, X.; Sun, Y.; Wang, K. Edge QoE: Computation Offloading with Deep Reinforcement Learning for Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 9255–9265. [CrossRef]
37. Motlagh, N.H.; Bagaa, M.; Taleb, T. UAV Selection for a UAV-Based Integrative IoT Platform. In Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016; pp. 1–6. [CrossRef]
38. Galkin, B.; Kibilda, J.; DaSilva, L.A. UAVs as Mobile Infrastructure: Addressing Battery Lifetime. *IEEE Commun. Mag.* **2019**, *57*, 132–137. [CrossRef]
39. Mekikis, P.-V.; Antonopoulos, A. Breaking the Boundaries of Aerial Networks with Charging Stations. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6. [CrossRef]
40. Mao, Y.; Zhang, J.; Song, S.H.; Letaief, K.B. Stochastic Joint Radio and Computational Resource Management for Multi-User Mobile-Edge Computing Systems. *IEEE Trans. Wirel. Commun.* **2017**, *16*, 5994–6009. [CrossRef]
41. Shannon, C.E. A mathematical theory of communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]

42. Nasir, Y.S.; Guo, D. Multi-Agent Deep Reinforcement Learning for Dynamic Power Allocation in Wireless Networks. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 2239–2250. [CrossRef]
43. Narendra, K.S.; Wang, Y.; Mukhopadhay, S. Fast Reinforcement Learning using multiple models. In Proceedings of the 2016 IEEE 55th Conference on Decision and Control (CDC), Las Vegas, NV, USA, 12–14 December 2016; pp. 7183–7188. [CrossRef]
44. Arulkumaran, N.K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Mag.* **2017**, *34*, 26–38. [CrossRef]
45. Kiran, B.R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A.A.; Yogamani, S.; Pérez, P. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 4909–4926. [CrossRef]
46. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.