*Article*

# DECCo-A Dynamic Task Scheduling Framework for Heterogeneous Drone Edge Cluster

**Zhiyang Zhang [1], Die Wu [1,2], Fengli Zhang [1] and Ruijin Wang [1,*]**

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
[2] Chengdu Aerospace Communication Device Company Limited, Chengdu 610052, China
* Correspondence: ruijinwang@uestc.edu.cn

**Abstract:** The heterogeneity of unmanned aerial vehicle (UAV) nodes and the dynamic service demands make task scheduling particularly complex in the drone edge cluster (DEC) scenario. In this paper, we provide a universal intelligent collaborative task scheduling framework, named DECCo, which schedules dynamically changing task requests for the heterogeneous DEC. Benefiting from the latest advances in deep reinforcement learning (DRL), DECCo autonomously learns task scheduling strategies with high response rates and low communication latency through a collaborative Advantage Actor–Critic algorithm, which avoids the interference of resource overload and local downtime while ensuring load balancing. To better adapt to the real drone collaborative scheduling scenario, DECCo switches between heuristic and DRL-based scheduling solutions based on real-time scheduling performance, thus avoiding suboptimal decisions that severely affect Quality of Service (QoS) and Quality of Experience (QoE). With flexible parameter control, DECCo can adapt to various task requests on drone edge clusters. Google Cluster Usage Traces are used to verify the effectiveness of DECCo. Therefore, our work represents a state-of-the-art method for task scheduling in the heterogeneous DEC.

**Keywords:** drone edge cluster; mobile edge computing; task scheduling; deep reinforcement learning

**Citation:** Zhang, Z.; Wu, D.; Zhang, F.; Wang, R. DECCo-A Dynamic Task Scheduling Framework for Heterogeneous Drone Edge Cluster. *Drones* **2023**, 7, 513. https://doi.org/10.3390/drones7080513

Academic Editors: Wenzheng Xu, Tang Liu and Weifa Liang

Received: 11 June 2023
Revised: 26 July 2023
Accepted: 1 August 2023
Published: 3 August 2023

## 1. Introduction

As the technologies of mobile edge computing (MEC) and unmanned aerial vehicles (UAVs) continue to evolve, there is an emergence of UAVs that support the Internet of Everything (IoE) and distributed learning services for artificial intelligence (AI) based on 6G are being widely applied in a plethora of sectors, including agriculture, logistics, photography, emergency response, and more [1,2]. In a drone edge cluster (DEC), UAVs act as edge nodes, processing tasks in close proximity to reduce network latency and augment data processing speed. These tasks have intensive resource requirements and handle large amounts of data by providing functions such as task scheduling, load balancing, task offloading and migration, and data privacy protection [3–5].

Collaborative task scheduling is a key process in the operation and management of DECs, which involves determining how to effectively allocate computing tasks to cluster nodes to optimize performance and resource utilization [6]. First, it decomposes complex, computationally intensive tasks into smaller, more manageable subtasks and places them in the task queue of the cluster. Secondly, the collaborative scheduling algorithm determines how to allocate tasks to different devices by obtaining the device status (available resources, network conditions, energy consumption, etc.) in the cluster and developing algorithms or strategies; Finally, the cluster collects execution results and dynamically adjusts task allocation and scheduling strategies to respond to changes in the cluster environment based on real-time feedback and monitoring data.

Typically, as shown in Figure 1, for a DEC consisting of a cloud, a base station issuing task instructions, and many UAVs, the base station must manage all service entities across the UAVs and the cloud while determining where to process these task requests. In industry, similar mobile edge clusters are often built on Kubernetes [7], aimed at seamlessly integrating distributed and layered computational resources on edge servers and terminal nodes [8]. In these existing distributed edge cluster frameworks, an effective coordinated scheduling algorithm should proactively schedule task requests based on resource utilization and task responsiveness within the DEC. There are two main reasons why these distributed architectures cannot be directly adapted to DEC scenarios:

1. Due to the different types and quantities of IoE services provided by UAV nodes, they show heterogeneity in terms of hardware resources (including CPU and memory, etc.) and network resources (including communication latency and data transmission latency, etc.). At the same time, UAV task scheduling also faces the limitations of MEC computing resources and the dynamic changes in different task requests. Some customized distributed edge computing frameworks (e.g., KubeEdge [9]) fail to match such dynamic, heterogeneous DEC scenarios;
2. The DEC collaborative task scheduling in the UAV scenario needs to comprehensively consider multiple indicators such as resource load balancing among UAVs, resource overload, and communication latency. The conventional edge computing frameworks usually only schedule for a single indicator, which makes them unable to completely describe the global state of DECs and local states of UAVs.

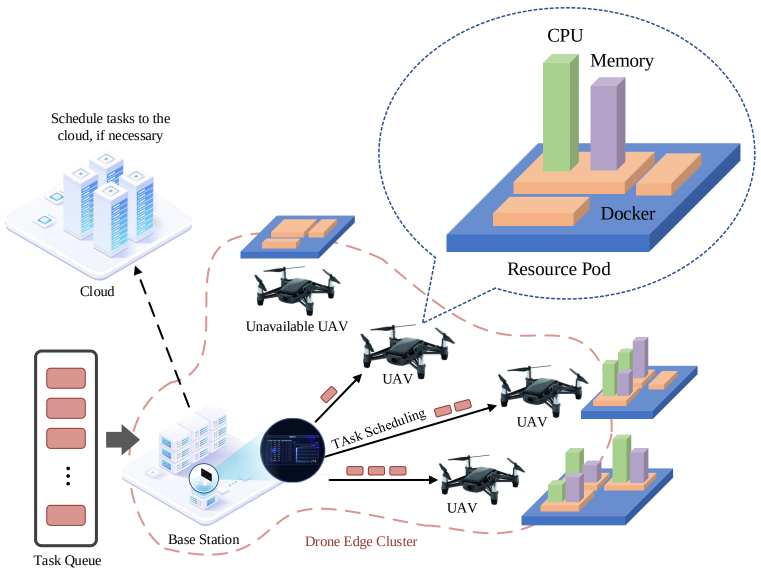


**Figure 1.** Drone Edge Cluster.

Several heuristic solutions have already been applied in the UAV-assisted mobile edge computing domain, including metaheuristic approaches [10], genetic algorithms (GA) [11], and particle swarm optimization (PSO) [12]. These algorithms focus on UAV trajectory planning and task offloading, requiring extensive iterations in stable environments to attain acceptable performance. In the case of mobile edge clusters, which undergo significant changes due to the fluctuation of available resources and request loads, these algorithms tend to generate frequent re-optimization and cause a considerable computational burden during iteration. Consequently, they must be more suited for the dynamic scheduling of DEC hosting IoE and AI services.

Some academic research has recently attempted to utilize reinforcement learning (RL) to formulate scheduling policies to support real-time DEC in dynamic environments [13]. RL obviates the need to solve optimization problems by exploring the dynamic environment of MEC and acquiring effective scheduling policies through experiential learning. However, with the increase in the number of UAVs, the state and action spaces of DEC will grow exponentially, significantly lowering the convergence efficiency of such methods. Deep reinforcement learning (DRL) schemes are designed to comprehend the natural environment of DEC through interaction and trial-and-error learning to tackle the issue of time complexity. With the aid of simulated environments and feedback mechanisms, UAVs autonomously learn and optimize decisions on flight, task allocation, and more. Each UAV can be considered an intelligent agent, whose behavioral policies are continually optimized through interactions with and feedback from the environment, enabling effective coordination among multiple UAVs [14–16]. In such algorithms/frameworks, the deep learning component handles the perception and prediction of the UAVs, while the reinforcement learning component takes charge of policy fine-tuning. Their combination is capable of managing complex and persistent cooperative scheduling tasks.

In summary, the variability of the task requests and cluster load often lead to severe scheduling errors or suboptimal decisions [15,17]. Therefore, relying solely on heuristic algorithms based on system dynamics assumptions or exploration-based model-free algorithms is impractical. Furthermore, scheduling tasks between UAVs with different resource availability can lead to local resource overload and downtime. The algorithm must consider cluster load balancing and downtime scenarios while implementing efficient, low-latency task scheduling.

To overcome these challenges and limitations, we introduce DECCo, a distributed edge computing cluster framework designed for UAV task collaborative scheduling scenarios. DECCo manages the service containers on each UAV, scheduling task requests for each UAV to achieve load balancing. The UAV coordination scheduling algorithm mA2C based on Advantage Actor–Critic is deployed in DECCo. The main contributions are summarized as follows:

1. We propose a distributed framework for DEC task scheduling, named DECCo. It can schedule dynamically changing task requests for heterogeneous DECs while ensuring high response rates and low communication latency. Tailored for real-world requirements, the architecture provides two complementary solutions for UAV collaborative scheduling, switching based on real-time scheduling performance;
2. We designed a targeted and scalable cost function for the collaborative scheduling of DEC, which includes aspects of resource load balancing, resource overload penalty, and communication latency optimization. Some custom objectives, such as priority and geographic distribution, can also be added to the cost function, so DECCo can adjust and generate required scheduling operations based on specific preferences related to the natural DEC environment;
3. We designed a UAV coordination scheduling algorithm mA2C based on Advantage Actor–Critic, which uses DRL to evaluate the global status of DEC and optimizes the UAV collaborative scheduling policy based on the policy gradient algorithm. mA2C uses target networks and experience replay to enhance the stability of scheduling policy learning, and avoids interference from local downtime on policy learning through action space masking.

In the sequel, Section 2 presents the work progress related to this paper, and Section 3 introduces the detailed design of the DECCo framework. Section 4 elaborates on the Markov process design and DRL algorithm design in DECCo. Section 5 presents the experiment results. Finally, Section 6 concludes the paper.

## 2. Related Work

### *2.1. Classical Solutions*

Classical solutions for DEC context have been explored, based on the joint optimization of request scheduling and service orchestration as proposed in [16,18]. Jeong et al. [19] addressed the joint optimization problem of bit allocation for task offloading in UAV MEC systems and achieved minimal energy consumption using a successive convex approximation strategy. Zhang et al. [20] and Liu et al. [21] further approximated the optimal solution for the total energy consumption of mobile UAVs using the successive convex approximation method combined with Lagrangian duality and decomposition iteration. Spatial search pruning algorithms were employed in [22] to find optimal edge servers for migration and expansion. Additionally, Chai et al. [23] used a parameter-adaptive differential evolution algorithm to optimize the number and placement of UAVs based on the positions and quantities of IoT devices.

Leading companies in the industry provide MEC task scheduling solutions such as Google GKE (Google Kubernetes Engine) [24], Azure AKS (Azure Kubernetes Service) [25], and AWS EKS (Amazon Elastic Kubernetes Service) [26]. The scalability of Kubernetes cluster scheduling tools primarily enables their availability. However, these scheduling tools and their extensions have significant limitations in the context of DEC, i.e, the Kubernetes cluster schedule relies heavily on heuristic-based solutions and manual configurations [27].

These heuristic-based approaches predominantly rely on centralized resource scheduling and assume accurate modeling or prediction of computational resources, network requirements, and processing times for specific requests, which may impact QoS in most UAV mobile edge computing scenarios.

### *2.2. Reinforcement Learning Solutions*

Offering a solution that surpasses traditional machine learning approaches, reinforcement learning, based on the Markov decision process (MDP), has shown potential in addressing these problems [28]. Reinforcement learning solutions can linearly and nonlinearly approximate the state-action value function of a system, making them better suited to adapt to the dynamic environment of mobile edge computing. Furthermore, they can learn without prior knowledge. Reinforcement-learning-based DEC task scheduling solutions originated from various areas, such as multi-user and multi-server MEC network resource management, computation resource expansion, wireless network security, and content caching [29]. Yang et al. [30] proposed a deep reinforcement learning-based method for task offloading in UAV-assisted edge computing networks, identifying the optimal task offloading decision for each UAV node using deep reinforcement learning. Liu et al. [31] introduced a behavior selection strategy based on dual deep Q networks (DQN) to minimize the cost of mobile edge computing systems considering QoS. Bi et al. [15] proposed LyDROO, which combines Lyapunov optimization and DRL to optimize task scheduling. Users can either schedule computational tasks to UAV nodes or offload tasks to edge servers. Hoang et al. [32] transformed the multi-stage problem of DEC task scheduling into a per-time-slot problem using Lyapunov optimization and solved it using a DRL-based algorithm. This framework enables simultaneous task offloading to macro base stations and MEC servers installed on UAVs, leveraging deep reinforcement-learning and integrated learning for parallel computing.

Based on DRL algorithms such as DQN, actor–critic (AC) [33], and policy gradient (PG) [34], some mobile edge cluster scheduling frameworks incorporate domain-specific knowledge for centralized scheduling. For example, Haja et al. [35] schedule services that satisfy service requests by periodically measuring the latency between edge nodes. Rossi et al. [36] utilizes model-based RL for service orchestration, leveraging the geographical distribution of edge nodes as expert knowledge. Kumar et al. [37] and Kim et al. [38] employ time-series forecasting methods to identify seasonal patterns and trends in task demands.

Compared with heuristic solutions, DRL-based solutions show advantages in the fields of complex scheduling space modeling, online learning and optimization, and uncertainty handling. However, such solutions need trial and error to learn the environmental changes of DEC. This can lead to erroneous scheduling decisions when faced with new task demand patterns. This is a common problem that needs to be addressed in all DRL-based solutions.

## 3. DECCo Framework Design

As shown in Figure 2, DECCo, which comprises the Cluster Orchestrator, Co-Scheduler, and the Cluster Controller, covers common DEC situations involving a base station and

UAVs. In this architecture, tasks arriving randomly at the base station carry demands of varying levels that dynamically change over time. Using the Co-Scheduler, tasks are deployed in the resource pool of UAVs in containers. When UAVs have sufficient available resources, this architecture can ensure high response rates and low communication latencies for task scheduling.

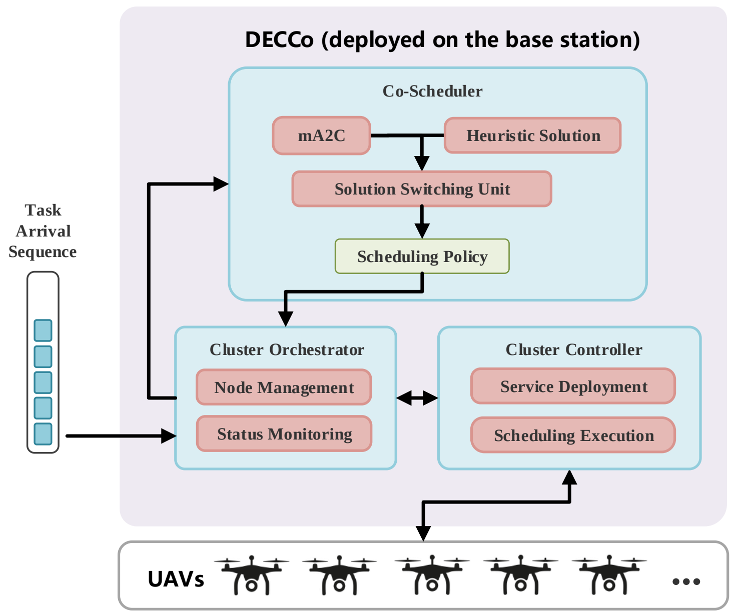


**Figure 2.** DECCo framework.

### 3.1. *Cluster Orchestrator*

The Cluster Orchestrator is responsible for the initialization, management, and configuration of UAVs, as well as real-time monitoring of global status information, including UAV resource usage, service deployment status, and connection status. When new tasks arrive, the Cluster Orchestrator is responsible for receiving these tasks and passing them to the Co-Scheduler. When the Co-Scheduler generates a new scheduling policy, the Cluster Orchestrator provides it to the Cluster Controller and keeps a record of these as logs. Serving as middleware, the Cluster Orchestrator connects the Co-Scheduler and the Cluster Controller.

### 3.2. *Co-Scheduler*

The Co-Scheduler, a DEC coordination scheduling solution based on DRL and heuristic algorithms, is responsible for assigning task requests arriving at the Cluster Orchestrator to available UAVs. The Co-Scheduler constructs a scheduling policy for the entire cluster to accommodate task requests with varying requirements and dynamically changing UAV hardware resources. The ultimate goal of a Co-Scheduler is to find the task scheduling policy $\pi$, which takes the cluster state as input and outputs actions that minimize future costs. Real-world task scheduling cannot tolerate errors or suboptimal decisions, which could lead to serious QoS and QoE implications, directly affecting the coordinated operation of UAVs. The Co-Scheduler executes policy improvement and avoids suboptimal decisions through the following three components.

#### 3.2.1. mA2C

mA2C is an intelligent scheduling solution based on DRL, interacting with the DEC through an improved actor–critic algorithm to handle the scheduling action space and balance dynamically changing system resource loads. mA2C combines experience replay with target networks and employs a policy gradient method to update the policy $\pi$, thus promoting successful training. mA2C is based on model-free DRL algorithms, which boast the advantage of intelligent and efficient learning. However, because mA2C learns the natural environment of DEC through interaction and trial and error, it may make suboptimal decisions in the initial phase of policy improvement or face different task request modes.

#### 3.2.2. Heuristic Solution

To address the above issue, a heuristic solution is also deployed to replace suboptimal decisions of mA2C during learning. In this paper, we set this solution as an evolutionary algorithm, which iteratively learns independently of mA2C and makes decisions on newly observed states or patterns during mA2C's learning process. The heuristic solution must wait for new task requests to arrive at the base station before making decisions, leading to no guarantee of its performance.

#### 3.2.3. Solution Switching Unit

This component compares and switches the outputs of the heuristic solution and mA2C. Specifically, the selection of scheduling actions starts with the output of the heuristic solution. When the count of more optimal decisions made by mA2C exceeds a predefined threshold, the solution switches to mA2C.

### *3.3. Cluster Controller*

The Cluster Controller is the component directly connected to UAVs. It executes task scheduling according to the scheduling operations provided by the Cluster Orchestrator. That is, the Cluster Controller allocates tasks to the corresponding UAVs based on the scheduling policy generated by the Co-Scheduler. In addition, the Cluster Controller is also responsible for deploying services and transmitting the status information of the UAVs to the Cluster Orchestrator.

## 4. Problem Formulation and System Model

### *4.1. Background*

In this paper, we consider a DEC system comprising a cloud, a base station, and the UAVs under its management, where the base station and UAVs are interconnected via a local area network (LAN). Upon the arrival of a task request at the base station, the latter assigns the task to the cloud or one of the UAV nodes it manages. Although a multi-tier DEC system can be envisaged—in which a cooperative DEC server is established for multiple base stations, thereby enabling collaboration among several base stations—this manuscript focuses on the scenario of coordinated task scheduling within a singular base station. The exemplification of our model will be based on this focused scenario in the subsequent sections. Table A1 summarizes the notations commonly used in Section 4.

1. Base Stations and UAVs: The UAVs are represented by the set $\mathcal{U} = \{1, 2, \ldots, U\}$, which refers to UAV nodes connected to and managed by the base station. Upon the arrival of a task request at the base station, it is either dispatched to the cloud or allocated to its supervised UAVs for processing. To facilitate the processing of task requests, each UAV should contain corresponding service entities;
2. Cloud: Unlike the UAVs within DEC, the cloud possesses abundant computational and storage resources. It is connected to the base station via a wide area network (WAN). When the UAVs cannot fulfill the tasks allocated by the base station, the cloud shoulders these tasks.

### *4.2. Scheduling Problem Formulation*

#### 4.2.1. MDP Formulation for DECCo

Task scheduling policy $\pi$ permits DEC to independently determine which UAV or cloud should provide services for incoming task requests. We formalize the process by which a DEC independently executes request scheduling as a Markov process (MDP), which forms the main framework for solving task scheduling policy $\pi$ using RL.

An MDP is represented by $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \mathcal{C}, \mathcal{P}r, \mathcal{Y})$. $\mathcal{S}$ is the state space, i.e., the collection of all possible states of the DEC. Within this state space, we encapsulate the dynamic changes in resource availability of each UAV under the control of the base station at any given time slot $t$. $\mathcal{C}$ is the cost function reflecting the agent's objective. $\mathcal{P}r$ is the probability transition matrix, outputting the probability distribution of entering the next state s, given the current state $s$ and action $a$. Finally, $\mathcal{Y}$ is the discount factor. In this paper, the elements of $(\mathcal{S}, \mathcal{A}, \mathcal{C}, \mathcal{P}r, \mathcal{Y})$ are defined as follows:

1. State Space. $\mathcal{S}$ denotes the state space. For each time slot $t$, we construct a state $s_t$ for the base station, which includes (i) the resource request $E_t = (E_t^{cpu}, E_t^{mem})$ of the current task $q_t$, where $E_t^{cpu}$ and $E_t^{mem}$ represent the CPU and memory requirements of the task, respectively; (ii) the task request queue information $Q_t$ awaiting scheduling at the base station; (iii) the available resources $A_{u,t} = (A_{u,t}^{cpu}, A_{u,t}^{mem})$ and the hardware resources $H_u = (H_u^{cpu}, H_u^{mem})$ of UAV $u$. It is worth noting that UAV $u$ 's hardware resources do not necessarily reflect the resources available for tasks, as there is a boundary $R_u$, i.e., $A_u$ cannot fall below $R_u$. System administrators set this boundary to reserve expansion space for other applications in the event of system overload; and (iv) communication latency $D_t$ between the base station and each UAV, with $[D_t]_0$ denoting the communication latency between the base station and the cloud.
2. Action Space. $\mathcal{A} = i_0^N$ denotes that the current task request is assigned for execution on UAV $i$. We permit the base station to schedule only one task request for all UAVs it manages in a time slot $t$. All actions of the scheduling policy $\pi$ are deterministic, meaning that if $a_t = u$, then the base station will schedule the current request to the UAV with id $u$ in time slot $t+1$.
3. Cost Function. DECCo uses the cost function $\mathcal{C}$ to calculate the cost of task scheduling action $a_t = u$. In Section 4.2.2, we discuss the three objectives that make up the cost function: load balancing, resource overload penalty, and communication latency.
4. Probability Transition. We denote $\mathcal{P}r(s_{t+1} \mid s_t, a_t) \in [0,1]$ as the probability of the base station transitioning from state $s_t$ to $s_{t+1}$ given a deterministic action $a_t$. In this paper, the probability of the base station issuing the scheduling action $a_t$ is represented as $\pi(a_t \mid s_t)$.
5. Discount Factor. $\mathcal{Y}$ is a decimal number in the range $[0,1]$, usually close to 1. $\mathcal{Y}$'s primary use is to expedite convergence by discounting the reward for the next state.

### 4.2.2. Cost Function

Our cost function can be expressed as $\mathcal{C} = \lambda_1 \cdot \mathcal{C}_1 + \lambda_2 \cdot \mathcal{C}_2 + \lambda_3 \cdot \mathcal{C}_3$, where $\lambda_1$, $\lambda_2$, and $\lambda_3$ represent the corresponding weights for the given three cost items. These weights are adjusted based on the properties of the DEC and the actual scenario, enabling our cost function to adapt to different scenarios and UAV equipment dynamically.

1. Load Balancing $\mathcal{C}_1$: Balancing CPU and memory resources across all UAVs ensures efficient utilization of resources and promotes system stability. This objective function aims to keep the load on the CPU and memory resources of UAVs as balanced as possible after execution of the schedule, that is, at time slot $t+1$. The standard deviation of the CPU resources load of all UAVs at time $t$ is given by Equation (1):

$$\mathcal{C}_1^{cpu} = \sqrt{\frac{1}{U}\sum_{i=1}^{U}(A_{i,t+1}^{cpu} - \frac{\sum_{j=1}^{U} A_{j,t+1}^{cpu}}{U})^2}, \tag{1}$$

$$A_{u,t+1}^{cpu} = \begin{cases} \frac{A_{u,t}^{cpu} - E_t^{cpu}}{H_u^{cpu}}, if\, A_{u,t}^{cpu} - E_t^{cpu} \geq 0, \\ 0, otherwise, \end{cases} \tag{2}$$

where $A_{u,t}^{cpu}$ and $E_{u,t}^{cpu}$ represent the available CPU resources of the current UAV $u$ and the CPU resources required to execute the current task, respectively. $H_u^{cpu}$ indicates the CPU size of UAV $u$. In the above cost function, we only consider the available resource of UAVs whose CPU and memory resources are not overloaded after task scheduling, that is, $A_{u,t}^{cpu} - E_t^{cpu} \geq 0$. For actions that cause resource overload on UAV $u$, we set the standard deviation of CPU and memory resources on $u$ to 0 and punish the action in the resource overload cost function.The above computation refers to the cost function for CPU load balancing, and the process of calculating the load balancing cost $\mathcal{C}_1^{mem}$ for memory is identical. Finally, $\mathcal{C}_1 = \mathcal{C}_1^{cpu} + \mathcal{C}_1^{mem}$.

2. Resource Overload Penalty $\mathcal{C}_2$: In this objective function, the base station is penalized for causing the resource requirements of the task request to exceed the available resources of the UAV. We represent this objective's cost as $\mathcal{C}_2^{cpu}$ and $\mathcal{C}_2^{mem}$, which calculates the resource overload cost of the CPU and memory on each UAV after performing the task scheduling action $a_t = u$. The calculation process of the CPU resource overload cost $O_{u,t+1}^{cpu}$ is represented by Equations (3) and (4):

$$O_{u,t+1}^{cpu} = \begin{cases} \frac{|A_{u,t}^{cpu} - E_t^{cpu}|}{H_u^{cpu}}, & if A_{u,t}^{cpu} - E_t^{cpu} < 0, \\ & 0, otherwise, \end{cases} \tag{3}$$

$$\mathcal{C}_2^{cpu} = k \sum_{i=1}^{U} O_{i,t+1}^{cpu}. \tag{4}$$

If the scheduling action overestimates the available resources of UAV $u$, the base station will return the proportion of overloaded resources for this; otherwise, the resource overload cost is 0, which means the base station will not be penalized for this action. We sum and multiply the possible resource overloads on each UAV by $k(k > 1)$ to achieve a stricter overload penalty, which is crucial for preventing node downtime, similar to $\mathcal{C}_1$, $\mathcal{C}_2 = \mathcal{C}_2^{cpu} + \mathcal{C}_2^{mem}$.

3. Communication Latency $\mathcal{C}_3$: Unlike most models with the same functionality, some custom cost items can be added to the cost function of DECCo. DECCo can leverage and adjust these custom targets based on specific preferences related to the actual DEC environment to perform smarter scheduling operations more closely aligned with actual needs. One of the potential objectives to be considered in this study is the minimization of communication latency between the base station and the selected UAVs. Upon obtaining the communication latency $D_t$ for each UAV and the base station at time $t$, the communication latency cost can be represented as $\mathcal{C}_3 = [D_t]_u$. Given the co-existence of minimizing communication latency and balancing load, DECCo can consider hardware resources (i.e., load balancing) and network resources (i.e., communication latency cost) in the DEC during policy learning. Therefore, the weights of $\mathcal{C}_1$ and $\mathcal{C}_3$ in the total cost function will depend on the importance of maintaining the two types of cluster resources.

### 4.3. mA2C

This section presents the UAV coordination scheduling algorithm, mA2C, that we designed based on the Advantage Actor–Critic, as illustrated in Figure 3. mA2C consists of two parts: (i) acquiring the state-value function of the DEC using a target network for self-improvement, and (ii) updating scheduling policy parameters based on the policy gradient, while concurrently avoiding the particular case of UAV node unavailability. To enhance the stability of scheduling policy learning, mA2C employs a target network and experience replay. Algorithm 1 shows the training process of mA2C.

**Algorithm 1** Training mA2C

```
Input: DEC environment
Output: Task scheduling policy $\pi_{\theta_p}$
1: Initialize the policy network $\pi_{\theta_p}$, the evaluation network $\mathbb{Q}(\cdot,\cdot;\theta_e)$ , target network $\mathbb{Q}(\cdot,\cdot;\theta_t)$, $\theta_t \leftarrow \theta_e$);
2: Initialize the state of the DEC $s_t$, the discount factor $\mathcal{Y}$ and experience pool $\mathcal{P}$;
3: for slot $t$ = 1, 2, ... do do
4:   while $Q_t \neq \emptyset$ do
5:     Compute UAV masking vector $F_t$;
6:     Get $a_t, \mathcal{C}$ using $\pi_{\theta_p}$;
7:     Store $(s_t, a_t, \mathcal{C}, s_{t+1})$ in $\mathcal{P}$;
8:     if $|\mathcal{P}| > \mathcal{P}_{max}$ then
9:       Pop out the oldest experience;
10:      Select random mini-batch experience from $\mathcal{P}$;
11:      (Base Station Critic) Update evaluation network using Equations (5) and (6) for each experience;
12:      (UAV Actor) Improve policy network $\pi_{\theta_p}$ using Equations (7) and (10);
13:    end if
14:    if $t$ % $N == 0$ then
15:      $\theta_t \leftarrow \theta_e$;
16:    end if
17:    $s_t = s_{t+1}$
18:  end while
19: end for
```

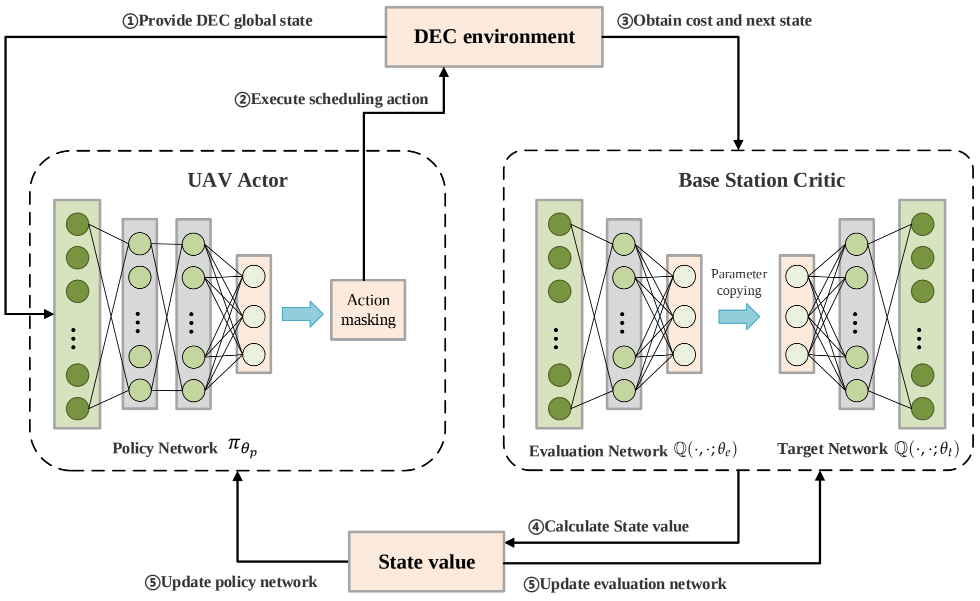

①Provide DEC global state
DEC environment
③Obtain cost and next state
②Execute scheduling action
UAV Actor
Action masking
Policy Network $\pi_{\theta_p}$
Base Station Critic
Parameter copying
Evaluation Network $\mathbb{Q}(\cdot,\cdot;\theta_e)$
Target Network $\mathbb{Q}(\cdot,\cdot;\theta_t)$
④Calculate State value
State value
⑤Update policy network
⑤Update evaluation network

**Figure 3.** mA2C.

### 4.3.1. Base Station Critic

The goal of the Base Station Critic is to solve for the state-value function of the base station. Initially, to represent the state-value function, we construct an evaluation network $\mathbb{Q}(\cdot,\cdot;\theta_e)$ and a target network $\mathbb{Q}(\cdot,\cdot;\theta_t)$. An estimate of the state value is then derived via the Bellman expectation equation:

$$v(s_{t+1};\theta_t) = \sum_{a_t} \pi(a_t|s_t)(r_{t+1} + \mathcal{Y}v(s_{t+1};\theta_t)), \tag{5}$$

Subsequently, the update of the evaluation network is performed by minimizing Equation (6):

$$L(\theta_e) = \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} [v(s_{t+1}; \theta_t) - v(s_t; \theta_e)]^2, \tag{6}$$

Here, $\mathcal{P}$ refers to the size of the experience pool, which is used to reduce the variance of the network parameter update and speed up convergence. During the network update process, we periodically copy the parameters of $\mathbb{Q}(\cdot, \cdot; \theta_e)$ to $\mathbb{Q}(\cdot, \cdot; \theta_t)$.

### 4.3.2. UAV Actor

We employ a policy gradient method with episode-based updates to refine the policy $\pi$, which can be represented by Equation (7):

$$\nabla_{\theta_p} J(\theta_p) = \psi_t \nabla_{\theta_p} ln \pi_{\theta_p}(a_t | s_t), \tag{7}$$

Here, $\theta_p$ refers to the parameters of the policy network $\pi_{\theta_p}$, while $\psi_t$ represents the estimate of the advantage function. $\psi_t$ is not restricted to a specific form. In this paper, we introduce a baseline function $B(s_t) = v(s_t; \theta_e)$, thus obtaining Equation (8):

$$\psi_t = [r_{t+1} + \mathcal{Y} v(s_{t+1}; \theta_t) - v(s_t; \theta_e)]^2. \tag{8}$$

Furthermore, in light of the complex realities of cooperative DEC environments, where networks frequently demonstrate disconnection, intermittent connectivity, and low-bandwidth situations, it is necessary to filter ineffective dispatch operations for the DEC to establish viable task scheduling targets.

Consider when UAV $u$ is unable to connect to the base station at time slot $t$, or when its communication latency $[D_t]_u$ with the base station surpasses a certain threshold $D_{max}$. In such scenarios, UAV $u$ can only execute stand-alone tasks such as returning to base or terminating cooperation. The data generated during this time will be cached locally and later uploaded to the edge or cloud when network connectivity is restored. Here, we maintain a binary vector $F_t\{0,1\}^{\{U+1\}}$ for the base station to count disconnected UAV nodes, where $[F_t]_u = 1$ implies that UAV $u$ is currently available, while $[F_t]_u = 0$ indicates its disconnected status. Notably, if $[F_t]_0$ equals 0, the base station cannot offload task requests to the cloud, which is a common situation often overlooked. Ultimately, we can optimize the original output $p(s_t)$ of the policy network $\pi_{\theta_p}$ according to $F_t$:

$$\hat{p}(s_t) = p(s_t) * F_t, \tag{9}$$

where $*$ stands for elementwise multiplication. The obtained $\hat{p}(s_t)$ is used to calculate $\pi_{\theta_p}(a_t|s_t)$ in Equation (7):

$$\pi_{\theta_p}(a_t = i \mid s_t) = \frac{[\hat{p}(s_t)]_i}{\|\hat{p}(s_t)\|}. \tag{10}$$

### 4.3.3. Heuristic Solution

Genetic algorithms are a class of evolutionary algorithms representing a global optimization technique that emulates natural selection and genetic mechanisms to search for solutions. In this study, we develop a heuristic solution for task scheduling in DECs based on genetic algorithms. It aims to learn the optimal task scheduling policies. It is worth noting that we utilize DECCo's cost function $\mathcal{C}$ in this algorithm.

The heuristic solution initially constructs a population of random task scheduling policies for a DEC. In each iteration of the algorithm, the performance of each policy is evaluated using the cost function $\mathcal{C}$, and the best-performing policies are selected for reproduction. During this process, the policies undergo crossover and mutation with specific probabilities. After a certain number of iterations, the heuristic solution returns the best-performing task-scheduling policy from the population as the output. As mentioned earlier, this output will be compared with the output from the mA2C in the Solution Switching Unit.

## 5. Experiments and Evaluations

### *5.1. Experiment Setup*

#### 5.1.1. Task Requests

We tailored and modified the Google Cluster Usage Tracing (GCT) [39] to fit our DEC environment. GCT provides extensive log data from Google's data centers, covering tasks performed in the data center and required CPU and memory resources over a period of time. We mapped machine units and task requests from the GCT dataset to our DEC collaborative task scheduling framework, DECCo, and used Co-Scheduler for task allocation.

#### 5.1.2. Cluster Construction

We have established an edge cluster on the Google Cloud Platform (GCP) [27] consisting of a master node (base station) and eight edge nodes (UAVs). Task requests randomly arrive at the master node and are distributed to the UAVs. The master node is configured with "2 vCPU, 4GB memory", while the four edge nodes are configured between "1 vCPU, 1–4 GB memory". Multiple "4 vCPU, 16 GB memory" virtual machines are configured in the cloud, and Linux TC controls communication latency with the edge cluster. For the simulation of unavailable UAV nodes, we deployed a probe similar to a latency detector on the master node. It will occasionally set the availability status of some nodes as unavailable and maintain this information in table form.

#### 5.1.3. Architecture Implementation

DECCo's Cluster Orchestrator regularly observes and collects the current edge cluster system state through a state monitor, including Docker services and physical nodes' hardware and network resources. For network resources, each edge node and the master node carry a latency detector for measuring communication latency. We deployed mA2C and heuristic solution proxy on the master node. mA2C and heuristic solution calculate scheduling actions by observing the global state in the state monitor. After solution-switching, the Cluster Orchestrator informs the Cluster Controller to execute this schedule for the current request.

### *5.2. Experimental Evaluation*

In our assessment, DECCo provides the DRL-based solution, mA2C, and the heuristic solution, MA. The solution switching unit in the framework determines which to use. mA2C is built on TensorFlow 2, wherein the evaluation and target networks implemented in Base Station Critic are DNNs with 256, 128, 64, and 32 neurons. UAV Actor is a three-layer deep neural network with 128, 64, and 32 hidden units per layer. The activation function of all networks is ReLU, and the learning rate is $10^{-3}$. For each cost goal described in the second section, we allocated weights: $\lambda_1 = 0.3; \lambda_2 = 0.5; \lambda_3 = 0.2$.

We compared our approach with several basic and advanced baselines, including the following:

1. Kubernetes scheduler [27]: This carries out scheduling operations based on system metrics monitored in real time. In this paper, we employ a purely greedy strategy, scheduling each request to the UAV node that minimizes load balancing and communication latency;
2. Heuristic solution: We directly demonstrate the independent decision making of the heuristic solution in DECCo, i.e., the evolutionary algorithm with cross-mutation;
3. LyDROO [15] and DRLRM [40]: These are both UAV scheduling algorithms based on Lyapunov optimization and DRL. The actor modules in them use DNNs and action quantizers to balance exploration and exploitation. The critic utilizes model-based optimization instead of deep neural networks. In this regard, LyDROO and DRLRM also combine intelligent and heuristic solutions. Due to different scheduling

scenarios, we set the Critic to a downright greedy strategy based on load balancing and communication latency minimization when using LyDROO and DRLRM as baselines. The Actor's network parameters are the same as mA2C.

### 5.2.1. Practicability of DECCo

To verify the applicability of DECCo under different task request modes, we trimmed the task demand queue into two modes: (i) random arrival mode, and (ii) a spliced mode with random arrival and CPU usage increment iteration under the same DEC environment. Figure 4 shows the convergence of DECCo in the task request random arrival mode, with time slots displayed on a logarithmic scale. Figures 5 and 6 compare the convergence of DECCo and other baseline algorithms/frameworks under the two task request modes. To highlight the convergence difference between DECCo and other baseline algorithms, we fitted the cost convergence curve with a polynomial to avoid the inevitable fluctuations during the convergence process.

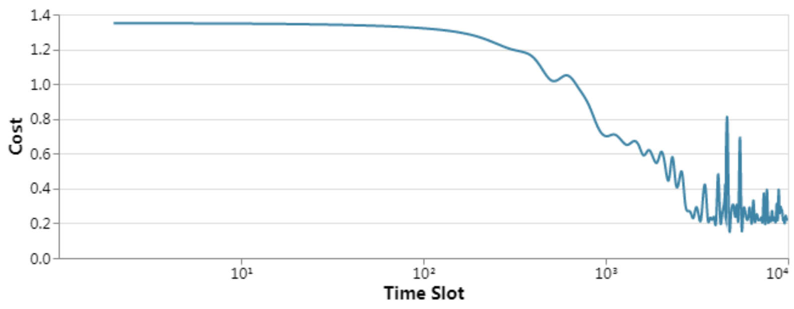
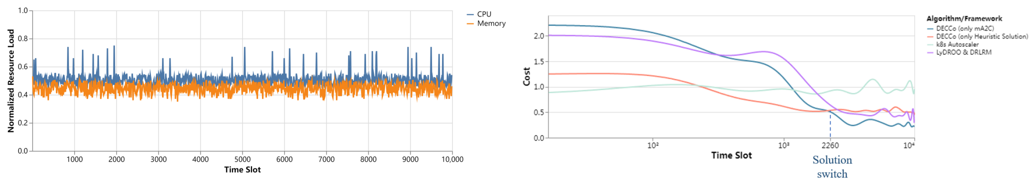


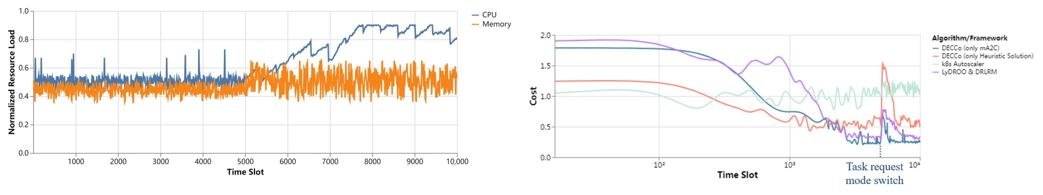
**Figure 4.** Practicability of DECCo.

In DECCo, the advantages of DRL-based solutions and heuristic solutions can be discerned from Figures 5 and 6. In the initial phase of agent learning (time slot 1 to $10^3$), the performance of the heuristic solution is significantly better than that of the DRL-based solution. This is mainly because mA2C needs some time to interact with the cluster environment and maintain some exploratory behavior, while the heuristic solution can make relatively accurate decisions after a small number of iterations. However, as the number of iterations increases, the performance of the DRL-based solution will be significantly better than that of the heuristic solution, determined by DRL's more vital convergence ability. Furthermore, when the agent faces unprecedented task request modes in the environment, the DRL-based solution will face the same problems as in the initial learning phase, leading to a transient increase in cost and a rapid decrease as the number of iterations increases. At this time, the heuristic solution can be a backup for mA2C to avoid relatively serious erroneous decisions in a short period. In the actual policy learning process, when mA2C outperforms the heuristic solution in 100 consecutive iterative evaluations, that is, at $t = 2260$, the solution switching unit will switch from using the heuristic solution to actively making decisions with mA2C, allowing DECCo to maintain the best decisions in the long term.



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

**Figure 5.** (**a**) Task request random arrival mode. (**b**) DECCo's performance in the task request random arrival mode.



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

**Figure 6.** (**a**) Task request splicing mode. (**b**) DECCo's performance in task request splicing mode.

### 5.2.2. Load Balancing

Figure 7 shows the impact of the allocation of weights $\lambda_1$, $\lambda_2$, $\lambda_3$ ($\lambda_1 + \lambda_2 + \lambda_3 = 1$) in the cost function's three objectives on load balancing. We use different $\lambda_1$, $\lambda_2$, and $\lambda_3$ values to train DECCo and calculate the convergence value of the load balancing goal $\mathcal{C}_1$, respectively. When "$\lambda_1 = 0.3, \lambda_2 = 0.5, \lambda_3 = 0.2$", DECCo shows the best performance. Furthermore, we focus on two parameter modes: (i) Excessive Attention to Load Balance. When "$\lambda_1 = 0.8, \lambda_2 = 0.1, \lambda_3 = 0.1$", the cost converges to 0.397, at this time DECCo overly focuses on load balancing, ignoring resource overload and network latency. (ii) Ignore Load Balancing. When "$\lambda_1 = 0, \lambda_2 = 0.1, \lambda_3 = 0.9$", the cost converges to 0.24; at this time, DECCo ignores load balancing and focuses on network latency between UAV and base station. At this time, DECCo's performance is not optimal but is better than the situation of overly focusing on load balance, indicating that overly focusing on load balance is not a good choice. In addition, if we increase the weight of the resource overload penalty $\lambda_2$ under the premise of ignoring load balancing (i.e., $\lambda_1 = 0$), the convergence value of $\mathcal{C}_1$ will decrease. This suggests that the overload penalty can also contribute to achieving load balance to some extent. A possible situation that may occur is that when the available resources of each drone node are close to zero, the model naturally achieves load balance while avoiding resource overload.

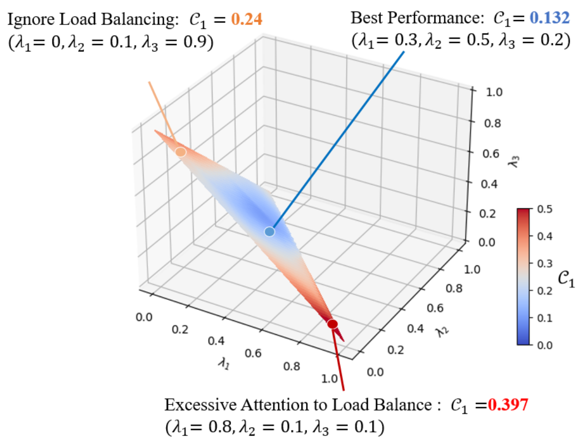


**Figure 7.** Impact of Cost Function Parameters on DECCo Performance.

### 5.2.3. Resource Overload

The resource usage of UAVs changes with the allocation of tasks. Therefore, the resource overload term in the cost function is used to penalize those operations that make the resource usage of UAVs exceed the resource boundary of UAVs. Since the resource demand of a single task is usually much smaller than the available resources of UAVs, resource overload only sometimes occurs. However, when the following two requirements are met, the loss of QoS caused by resource overload will far exceed the imbalance of load and high communication latency, and this loss is unpredictable: (i) the cluster resources are highly saturated; and (ii) there is a significant difference in the task-carrying capacity of UAVs.

To highlight the value of the resource overload penalty, we set an available resource boundary for each UAV. Specifically, we assign the same resource boundary value

$R_1 = R_2 = R_3 = R_4 = 0.8$ GB to four UAVs with memory of 1 GB, 2 GB, 2 GB, and 4 GB, respectively. In Figure 8, we show how the four UAVs' usage changes with the time slot. It can be seen that UAV 2 and UAV 3 experience resource overload at two time points. DECCo can effectively recover after a resource overload occurs and can converge the resource usage to a stable range. It is worth noting that, compared with UAV 3, the resource utilization rate on UAV 2 is higher because UAV 2 has a smaller communication latency than the base station. Because of the significant impact, such as node shutdown caused by resource overload to the cluster, we recommend setting $\lambda_2$ to a larger value.

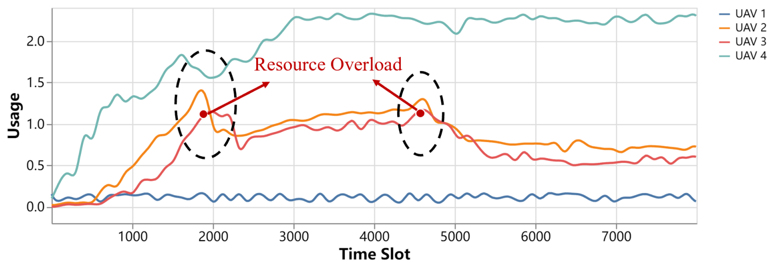


**Figure 8.** Impact of resource overload on UAV resource usage.

## 6. Conclusions

In this paper, we propose a smart collaborative task scheduling framework for drone edge clusters, DECCo, which autonomously learns task scheduling strategies with high response rates and low communication latencies through a cooperative Advantage Actor–Critic algorithm, ensuring load balance while avoiding resource overload and local shutdown interference during the task scheduling strategy learning process. DECCo avoids serious errors and suboptimal decisions by utilizing a scheduling solution switch unit, and adapts to various types of task requests on different DECs through flexible parameter control. We verified the effectiveness of DECCo through simulations based on a real-world cluster resource request dataset.

We plan to extend DECCo to cover microservice deployment and resource scheduling scenarios for DECs. This extension will not merely limit the action space of the Markov process within DECCo to the selection of UAV nodes. The scheduling framework will also need to consider how to efficiently deploy and schedule CPU and memory resources for the same task across different UAV nodes. This approach will further enhance the versatility and autonomous learning ability of DECCo across a range of DEC scenarios.

**Author Contributions:** Conceptualization, Z.Z.; Methodology, Z.Z., D.W., F.Z. and R.W.; Validation, Z.Z. and D.W.; Formal analysis, Z.Z.; Investigation, Z.Z.; Data curation, Z.Z.; Writing—original draft, D.W. and F.Z.; Writing—review & editing, Z.Z. and R.W.; Visualization, Z.Z.; Supervision, F.Z.; Project administration, D.W. and R.W.; Funding acquisition, R.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was partially supported by the National Natural Science Foundation of China under Grant 62271128, and the Sichuan Science and Technology Program Key R&D Project under Grant 2022ZDZX0004, 23ZDYF0706, 23ZDYF0085, 2022YFG0212, 2021YFS0391, 2021YFG0027

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

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

## Appendix A

**Table A1.** Notations and Explanations.

| Notation | Explanation |
|---|---|
| $\mathcal{U}$ | UAV nodes set |
| $q_t$ | Tasks scheduled by DEC at time slot $t$ |
| $E^t$ | The resource request of the current task $q_t$ |
| $A_u^t$ | The available resources of UAV $u$ at time slot $t$ |
| $H_u$ | The hardware resources of UAV $u$ |
| $D_t$ | Communication latency between the base station and each UAV at time slot $t$ |
| $s_t$ | The state of the DEC at time slot $t$ |
| $a_t$ | The scheduling action performed by DEC at time slot $t$ |
| $\mathcal{C}$ | Cost Function |
| $\mathcal{Y}$ | Discount factor |
| $\theta_e, \theta_t, \theta_p$ | Parameters of evaluation network, target network, policy network |
| $v(t)$ | The state value function of DEC at time slot $t$ |
| $\pi$ | The task scheduling policy |
| $F_t$ | Vector to record available UAV nodes |
| $\mathcal{P}$ | The size of the experience pool |
| $N$ | Target network update period |

## References

1. Giordani, M.; Polese, M.; Mezzavilla, M.; Rangan, S.; Zorzi, M. Toward 6G Networks: Use Cases and Technologies. *IEEE Commun. Mag.* **2020**, *58*, 55–61. [CrossRef]
2. Alameddine, H.A.; Sharafeddine, S.; Sebbah, S.; Ayoubi, S.; Assi, C. Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 668–682. [CrossRef]
3. Nath, S.; Wu, J. Deep Reinforcement Learning for Dynamic Computation Offloading and Resource Allocation in Cache-Assisted Mobile Edge Computing Systems. *Intell. Converg. Netw.* **2020**, *1*, 181–198.
4. Yang, T.; Hu, Y.; Gursoy, M.C.; Schmeink, A.; Mathar, R. Deep Reinforcement Learning Based Resource Allocation in Low Latency Edge Computing Networks. In Proceedings of the 2018 15th International Symposium on Wireless Communication Systems (ISWCS), Lisbon, Portugal, 28–31 August 2018; pp. 1–5.
5. Yang, L.; Yao, H.; Wang, J.; Jiang, C.; Benslimane, A.; Liu, Y. Multi-UAV-Enabled Load-Balance Mobile-Edge Computing for IoT Networks. *IEEE Internet Things J.* **2020**, *7*, 6898–6908. [CrossRef]
6. Tuli, S.; Ilager, S.; Ramamohanarao, K.; Buyya, R. Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks. *IEEE Trans. Mob. Comput.* **2020**, *21*, 940–954. [CrossRef]
7. Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. Borg, Omega, and Kubernetes. *Commun. ACM* **2016**, *59*, 50–57.
8. Wang, X.; Han, Y.; Leung, V.C.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutorials* **2020**, *22*, 869–904. [CrossRef]
9. Xiong, Y.; Sun, Y.; Xing, L.; Huang, Y. Extend Cloud to Edge with KubeEdge. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Bellevue, WA, USA, 25–27 October 2018; pp. 373–377.
10. Pham, Q.V.; Mirjalili, S.; Kumar, N.; Alazab, M.; Hwang, W.J. Whale Optimization Algorithm with Applications to Resource Allocation in Wireless Networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 4285–4297.
11. Tran, T.X.; Pompili, D. Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE Trans. Veh. Technol.* **2018**, *68*, 856–868. [CrossRef]
12. Li, B.; Fei, Z.; Zhang, Y. UAV Communications for 5G and Beyond: Recent Advances and Future Trends. *IEEE Internet Things J.* **2018**, *6*, 2241–2263. [CrossRef]
13. Huang, L.; Bi, S.; Zhang, Y.J.A. Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2019**, *19*, 2581–2593. [CrossRef]
14. He, Y.; Zhang, Z.; Yu, F.R.; Zhao, N.; Yin, H.; Leung, V.C.; Zhang, Y. Deep-Reinforcement-Learning-Based Optimization for Cache-Enabled Opportunistic Interference Alignment Wireless Networks. *IEEE Trans. Veh. Technol.* **2017**, *66*, 10433–10445. [CrossRef]
15. Bi, S.; Huang, L.; Wang, H.; Zhang, Y.J.A. Lyapunov-Guided Deep Reinforcement Learning for Stable Online Computation Offloading in Mobile-Edge Computing Networks. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7519–7537. [CrossRef]
16. Farhadi, V.; Mehmeti, F.; He, T.; La Porta, T.F.; Khamfroush, H.; Wang, S.; Poularakis, K. Service Placement and Request Scheduling for Data-Intensive Applications in Edge Clouds. *IEEE/ACM Trans. Netw.* **2021**, *29*, 779–792. [CrossRef]
17. Mao, H.; Alizadeh, M.; Menache, I.; Kandula, S. Resource Management with Deep Reinforcement Learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks, Atlanta, GA, USA, 9–10 November 2016; pp. 50–56.

18. Ma, X.; Zhou, A.; Zhang, S.; Wang, S. Cooperative Service Caching and Workload Scheduling in Mobile Edge Computing. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 2076–2085.
19. 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.* **2017**, *67*, 2049–2063. [CrossRef]
20. Zhang, T.; Xu, Y.; Loo, J.; Shen, X.S.; Wang, J.; Liang, Y.; Wang, X. Joint Computation and Communication Design for UAV-Assisted Mobile Edge Computing in IoT. *IEEE Trans. Ind. Inform.* **2019**, *16*, 5505–5516. [CrossRef]
21. Liu, Y.; Xiong, K.; Ni, Q.; Fan, P.; Letaief, K.B. UAV-Assisted Wireless Powered Cooperative Mobile Edge Computing: Joint Offloading, CPU Control, and Trajectory Optimization. *IEEE Internet Things J.* **2019**, *7*, 2777–2790. [CrossRef]
22. Li, C.; Sun, H.; Chen, Y.; Luo, Y. Edge Cloud Resource Expansion and Shrinkage Based on Workload for Minimizing the Cost. *Future Gener. Comput. Syst.* **2019**, *101*, 327–340. [CrossRef]
23. Chai, X.; Zheng, Z.; Xiao, J.; Yan, L.; Qu, B.; Wen, P.; Wang, H.; Zhou, Y.; Sun, H. Multi-Strategy Fusion Differential Evolution Algorithm for UAV Path Planning in Complex Environment. *Aerosp. Sci. Technol.* **2022**, *121*, 107287. [CrossRef]
24. Google Kubernetes Engine (GKE). Available online: https://cloud.google.com/kubernetes-engine (accessed on 1 August 2023).
25. Azure Kubernetes Service (AKS). Available online: https://azure.microsoft.com/en-us/services/kubernetes-service (accessed on 1 August 2023).
26. Amazon Elastic Kubernetes Service (EKS). Available online: https://aws.amazon.com/eks (accessed on 1 August 2023).
27. Google Cloud Platform (GCP). Available online: https://console.cloud.google.com (accessed on 1 August 2023).
28. Li, M.; Gao, J.; Zhao, L.; Shen, X. Deep Reinforcement Learning for Collaborative Edge Computing in Vehicular Networks. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *6*, 1122–1135. [CrossRef]
29. Luong, N.C.; Hoang, D.T.; Gong, S.; Niyato, D.; Wang, P.; Liang, Y.C.; Kim, D.I. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Commun. Surv. Tutor*. **2019**, *21*, 3133–3174. [CrossRef]
30. Yang, C.; Liu, B.; Li, H.; Li, B.; Xie, K.; Xie, S. Learning Based Channel Allocation and Task Offloading in Temporary UAV-Assisted Vehicular Edge Computing Networks. *IEEE Trans. Veh. Technol.* **2022**, *71*, 9884–9895. [CrossRef]
31. Liu, Q.; Shi, L.; Sun, L.; Li, J.; Ding, M.; Shu, F. Path Planning for UAV-Mounted Mobile Edge Computing with Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2020**, *69*, 5723–5728. [CrossRef]
32. Hoang, L.T.; Nguyen, C.T.; Pham, A.T. Deep Reinforcement Learning-Based Online Resource Management for UAV-Assisted Edge Computing with Dual Connectivity. *IEEE/ACM Trans. Netw.* **2023**. [CrossRef]
33. Grondman, I.; Busoniu, L.; Lopes, G.A.; Babuska, R. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Trans. Syst. Man Cybern. C Appl. Rev.* **2012**, *42*, 1291–1307. [CrossRef]
34. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Adv. Neural Inf. Process. Syst.* **1999**, *12*.
35. Haja, D.; Szalay, M.; Sonkoly, B.; Pongracz, G.; Toka, L. Sharpening Kubernetes for the Edge. In Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, Beijing China, 19–23 August 2019; pp. 136–137.
36. Rossi, F.; Cardellini, V.; Presti, F.L.; Nardelli, M. Geo-Distributed Efficient Deployment of Containers with Kubernetes. *Comput. Commun.* **2020**, *159*, 161–174. [CrossRef]
37. Kumar, J.; Singh, A.K.; Buyya, R. Self Directed Learning Based Workload Forecasting Model for Cloud Resource Management. *Inf. Sci.* **2021**, *543*, 345–366. [CrossRef]
38. Kim, I.K.; Wang, W.; Qi, Y.; Humphrey, M. Forecasting Cloud Application Workloads with Cloudinsight for Predictive Resource Management. *IEEE Trans. Cloud Comput.* **2020**, *10*, 1848–1863. [CrossRef]
39. Verma, A.; Pedrosa, L.; Korupolu, M.; Oppenheimer, D.; Tune, E.; Wilkes, J. Large-Scale Cluster Management at Google with Borg. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux France, 21–24 April 2015; pp. 1–17.
40. Zhu, X.; Luo, Y.; Liu, A.; Xiong, N.N.; Dong, M.; Zhang, S. A Deep Reinforcement Learning-Based Resource Management Game in Vehicular Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 2422–2433. [CrossRef]