electronics

Article

Federated Deep Reinforcement Learning for Joint AeBSs Deployment and Computation Offloading in Aerial Edge Computing Network

Lei Liu [1,*], Yikun Zhao [2], Fei Qi [1], Fanqin Zhou [2], Weiliang Xie [1], Haoran He [2] and Hao Zheng [2]

1 Beijing Research Institute, China Telecom Corporation Limited, Beijing 102209, China
2 State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
* Correspondence: liulei6@chinatelecom.cn

Abstract: In the 6G aerial network, all aerial communication nodes have computing and storage functions and can perform real-time wireless signal processing and resource management. In order to make full use of the computing resources of aerial nodes, this paper studies the mobile edge computing (MEC) system based on aerial base stations (AeBSs), proposes the joint optimization problem of computation the offloading and deployment control of AeBSs for the goals of the lowest task processing delay and energy consumption, and designs a deployment and computation offloading scheme based on federated deep reinforcement learning. Specifically, each low-altitude AeBS agent simultaneously trains two neural networks to handle the generation of the deployment and offloading strategies, respectively, and a high-altitude global node aggregates the local model parameters uploaded by each low-altitude platform. The agents can be trained offline and updated quickly online according to changes in the environment and can quickly generate the optimal deployment and offloading strategies. The simulation results show that our method can achieve good performance in a very short time.

Keywords: aerial network; mobile edge computing; 6G; computation offloading

check for updates

Citation: Liu, L.; Zhao, Y.; Qi, F.; Zhou, F.; Xie, W.; He, H.; Zheng, H. Federated Deep Reinforcement Learning for Joint AeBSs Deployment and Computation Offloading in Aerial Edge Computing Network. *Electronics* **2022**, *11*, 3641. https://doi.org/10.3390/electronics11213641

Academic Editors: Yichuang Sun, Haeyoung Lee and Oluyomi Simpson

Received: 26 September 2022
Accepted: 31 October 2022
Published: 7 November 2022

1. Introduction

Due to the explosive development of various intelligent applications in the 6G era, user's demand for computing and communication is expected to increase dramatically. To meet this challenge, mobile edge computing (MEC) is considered as an efficient paradigm, which can improve network computing capability and user experience [1]. At the same time, the aerial network based on new mobile communication systems such as high-altitude platforms (HAPs) is considered as a potential architecture for 6G and has aroused much attention recently. The aerial network acts as a supplement and extension of the terrestrial mobile communication network to provide collaborative and efficient information services for various network applications in a wide spatial area [2]. The aerial network primarily consists of low-altitude platforms and high-altitude platforms and plays a key role in enhancing coverage, enabling edge services, and enabling flexible network reconfiguration. Communication, computing, caching, sensing, and navigation services will be possible on a global scale through the fusion of aerial networks and edge computing [3].

Nevertheless, there are numerous obstacles to integrating aerial networks with MEC systems. A classic MEC system typically deploys edge servers using the ground infrastructure. The aerial base station (AeBS) can be formed by carrying the MEC server and other communication equipment on the aerial platform. An aerial edge computing node differs from its terrestrial counterpart, since it is capable of flexibly adjusting its deployment position to achieve better communication conditions [4]. The traditional offloading strategy for terrestrial MEC networks may not be appropriate for aerial edge computing. In addition, in order to extend the coverage or increase the number of users served, a single AeBS is not feasible and multiple AeBSs can be cooperatively deployed to complete the task [5]. How to coordinate the deployment of multiple AeBSs is also a critical issue. On the other hand, although the flexibility of AeBSs brings a higher degree of freedom to the deployment design of MEC nodes, they often suffer from resource and energy constraints. Therefore, how to make the AeBSs better provide communication and computing services under the condition of limited resources is also a problem that needs to be solved [1].

Edge computing and deep learning can naturally be combined with each other. On the one hand, they complement each other technically [6]. Some recent work has focused on the generalization problem of deep neural network (DNN) models [7] or designing resource-friendly models [8,9] to help the DNN model be better applied in actual edge deployment scenarios. On the other hand, their application and popularization are mutually beneficial. The combination of the two technologies has promoted a wide range of intelligent applications, from face recognition [10] and drone navigation [11] to the Industrial Internet of Things [12]. Deep learning has strong perception and expression ability, while reinforcement learning has decision-making ability. Deep reinforcement learning (DRL), which introduces deep neural networks in deep learning into reinforcement learning, holds promise for generating network optimization decisions in complex and dynamic environments.

Traditional single-agent DRL approaches usually follow a centralized paradigm, performing poorly in scalability and flexibility due to their large state space and action space [13]. Recently, researchers have discovered that multi-agent deep reinforcement learning (MADRL), which handles modeling and computation in a distributed manner, can obtain better performance in solving multi-AeBS cooperation tasks [14]. However, in MADRL, agents need to interact with each other to exchange state and action information in order to maintain the stability of the environment. In practical situations, frequent communication between AeBSs will consume the communication resources of AeBSs and increase the complexity of the optimization problem. As a distributed training framework, federated learning can simplify the model and improve the convergence speed in large-scale MADRL models. The benefits of applying federated learning to MADRL can be mainly summarized as follows: (1) federated learning avoids direct data leakage and, thus, can protect data privacy [15,16], (2) federated learning can make DRL models converge quickly, so it performs well in some scenarios sensitive to model training time [17], (3) incorporating federated learning into DRL can improve system scalability [18], (4) federated learning can also address the data island problem [16,18].

In this paper, we focus on the joint optimization of a mobile device (MD) offloading scheme and AeBS deployment to fully utilize the resources of AeBSs. Considering the dynamic nature of communication networks and computational tasks, we propose a federated DRL-based scheme that simultaneously addresses AeBS deployment and MD computational offloading. The contributions of our work can be concluded as:

1. A federated DRL algorithm was designed to jointly optimize the AeBSs' deployment and computation offloading to achieve lower energy consumption and task processing delay.
2. A new training mechanism is presented in the aerial edge computing network where low-altitude AeBSs are controlled by their own agents and cooperate in a distributed manner, and an HAP acts as a global node for model aggregation to improve the training efficiency.
3. Two neural networks trained together were set up for each agent to deploy the AeBSs and generate the computation offloading policies, respectively.

The content of this paper is organized as follows. Section 2 introduces some related work. Section 3 shows the system model of the aerial edge computing network and analyzes the joint deployment and offloading optimization problem that needs to be addressed. Section 4 describes the detailed flow and architecture of our proposed federated deployment and computational offloading (FedDCO) algorithm. Section 5 presents the results and analysis of our simulation for FedDCO. Section 6 is a summary of the full paper.

2. Related Work

Several works on integrating edge computing into aerial networks have been conducted. In [3], the survey introduced several desirable attributes and enabling technologies of aerial computing. In [19], Jia et al. studied the offloading problem in a hierarchical aerial computing framework composed of HAPs and AeBSs, and the flexible mobility of AeBSs was ignored. Reference [20] adopted an AeBS to provide computation offloading services for mobile users, but they did not study the dynamic computation offloading strategy. In [21], Truong et al. investigated an aerial edge computing network where an HAP plays the role of MEC and an offloading optimization problem is formulated, aiming to minimize the cost for task completion.

Researchers have adopted some heuristic algorithms to solve the offloading decision problem. To increase the system's weighted computing efficiency, Reference [4] proposed a heuristic algorithm for maximizing computational efficiency. In order to reduce the energy used by the AeBSs, Reference [22] jointly optimized the offloading of computation bits and the trajectory of the AeBS in an AeBS-enabled MEC system. In Reference [19], a matching game-theory-based algorithm and a heuristic approach for offloading optimization were presented. However, considering the dynamic nature of the multi-AeBS scenario, network optimization decisions are expected to be real-time. These aforementioned algorithms usually take many iterations to reach a local optimum, which makes them unsuitable for practical computation offloading situations. Besides, their computational complexity tends to rise significantly with the expansion of MEC network scale.

Recently, deep learning has made a series of achievements in the field of wireless communication, and researchers have also investigated some advanced models to help the deep neural network be better applied in practice. To address the generalization issue of the deep neural network, a two-stage training method was devised to optimize the feature boundary of the convolution neural network (CNN) to reduce the over-fitting problem in [7]. Several works were devoted to designing a resource-friendly edge artificial intelligence model. Reference [8] designed a graphics processing unit (GPU)-accelerated faster mean-shift algorithm, which is valuable for accelerating the speed of the training of the DNN model and saving computing resources. Reference [9] implemented a classification system based on the multi-channel CNN, which can work in a hardware environment with limited computing resources. Some researches also discussed the application of deep learning in MEC networks. Reference [23] utilized distributed deep learning to make offloading decisions for MEC networks in parallel. Reference [24] developed a hierarchical deep learning task distribution framework to deal with the tradeoff between latency and energy consumption, where the unmanned Aerial Vehicles are embedded with lower layers of the pretrained CNN model, while the MEC server handles the higher layers. These studies demonstrate the potential of combining deep learning with edge computing and also reveal the importance of generalization and resource issues in practical applications.

DRL, a combination of the DNN and reinforcement learning, aims to create an intelligent agent that can execute effective strategies and maximize the return of long-term tasks through controllable actions. Reference [25] designed a fast deep-Q-network (DQN)-based offloading approach to boost computation performance. Reference [26] provided a DQN-based online computation offloading policy with random task arrivals in a similar network setup. The authors in [27] adopted a centralized DRL algorithm to settle the offloading issue and a differential-evolution-based approach to address the deployment issue. The optimization issue of maximizing the migration throughput of user workloads in aerial MEC systems was solved with a DRL method in [28]. Reference [21] utilized the deep deterministic policy gradient (DDPG) to reduce the overall cost of performing the tasks. DRL was also used to jointly tackle the optimization problem of user association and resource allocation in aerial edge computing systems [29].

MADRL has numerous advantages over single-agent DRL. It enables agents to work cooperatively to handle high-complexity tasks in a distributed manner. Different MADRL algorithms have different agent-to-agent interaction forms and communication costs. A multi-agent imitation learning technique was presented in [30] to reduce the average task completion time in edge computing networks. To reduce overall energy usage, Reference [31] employed a multi-agent path planning strategy for energy consumption minimization. Reference [32] devised an MADRL-based trajectory control approach, which plans the trajectory of each AeBS individually. To reduce the overall computation and communication cost, Reference [33] developed a decentralized value-iteration-based reinforcement learning approach to make joint computation offloading and resource allocation decisions. The above research discovered that the multi-agent algorithm performs effectively in the multi-AeBS control scenario. This is because the MADRL framework considers the system as a whole and can jump out of the local optimal solution, which maximizes the benefit of each agent.

Some researchers have introduced federated learning into the DRL algorithm. Federated learning can accelerates the convergence speed of the model and enhances the generalization ability of the model by aggregating parameters. Reference [34] jointly optimized resource allocation, user association, and power control in a multi-AeBS MEC system via a federated DQN approach. In a multi-AeBS MEC system, massive amounts of data have to be transmitted from UEs to the parameter center. The practical deployment and operation of the algorithm are challenging because of the corresponding communication delay. To solve this problem, the authors fused federated learning (FL) with the MADRL framework and proposed a semi-distributed multi-agent federated reinforcement learning algorithm with the integration of FL and DRL. The proposed algorithm enables the UEs to quickly learn models by keeping their data training locally. In [35], an edge federated multi-agent actor–critic approach for resource management, collaborative trajectory planning, and data scheduling was provided. For cooperation in MEC systems, a federated heterogeneous multi-agent actor–critic algorithm was designed in [36]. Reference [37] designed a federated DRL-based cooperative edge caching architecture, which enables base stations to cooperatively learn a shared model and addresses the complicated and dynamic control concerns. A hierarchical federated DRL approach was described in [38] in a content replacement scenario.

The aforementioned works inspired us to design a federated deep reinforcement learning algorithm in which each AeBS is managed by a separate agent and cooperates in a distributed way, aiming to reduce the overall task processing time and energy consumption. Table 1 lists a comparison of our work to the relevant research.

Table 1. Comparison between our work and the existing literature.

Reference	Optimization Goal	Offloading	Deployment	Method
[4]	Maximize the weighted computational efficiency of the system	Proportional	✓	Alternative computational efficiency maximization
[19]	Maximize the total IoT data computed by the aerial MEC platforms	Binary	/	Matching game-theory-based algorithm and a heuristic algorithm
[20]	Minimize the total energy consumption	/	✓	Successive convex approximation (SCA)
[21]	Minimize the total cost function of the system	Proportional	/	Deep deterministic policy gradient (DDPG)
[22]	Minimize the energy consumed at the AeBS	Binary	✓	Alternative optimization

Table 1. *Cont.*

Reference	Optimization Goal	Offloading	Deployment	Method
[23]	Minimize overall system utility including both the total energy consumption and the delay in finishing the task	Binary	✓	DNN
[24]	Minimize the delay and energy consumption, while considering the data quality input into the DNN and inference error	Binary and proportional	/	CNN
[25]	Optimal offloading policy	Proportional	/	Fast deep-Q-network (DQN)
[26]	Maximize the long-term utility performance	Binary	/	Double DQN
[27]	Average slowdown for offloaded tasks	One-to-one correspondence	✓	DQN
[28]	Maximize the migration throughput of user tasks	Binary	/	DQN
[29]	Maximize the average throughput of user tasks	Binary	✓	Q-learning
[30]	Minimize average task completion time	Binary	/	Multi-agent imitation learning
[31]	Minimize the total energy consumption of AeBSs	Binary	✓	Multi-agent deep deterministic policy gradient (MADDPG)
[32]	Maximize the fairness among all the user equipment (UE) and the fairness of the UE load of each AeBS	Binary	✓	MADDPG
[33]	Minimize the total computation and communication overhead of the joint computation offloading and resource allocation strategies	Binary	/	Multi-agent double-deep Q-learning
[34]	Minimize the overall consumed power	Binary	/	Federated DQN
[35]	Minimize the average source age (elapsed time)	Binary	✓	Federated multi-agent actor–critic
[36]	Minimize the average age of all data sources	Binary	✓	Federated multi-agent actor–critic
[37]	Maximize the expected long-term reward	Three-way	✓	Federated DQN
[38]	Improve the hit rate	Binary	/	Federated DQN
Our work	Jointly minimize overall task latency and energy consumption	Binary	✓	Federated DQN

Notes: Binary: tasks can be offloaded to the AeBSs or not; proportional: tasks have the offloading rate, and tasks can be offloaded partially; one-to-one correspondence: tasks must be offloaded to an associated AeBS; three-way: tasks can be offloaded to the AeBSs, processed locally, or processed by their neighbors.

3. System Model

Figure 1 depicts our scenario in an aerial edge computing network. There are M MDs, N low-altitude AeBSs, and an HAP in the system. For simplicity, we abbreviate low-altitude AeBSs as AeBSs and consider all AeBSs and MDs to be computationally capable, and additionally, we used a binary offloading strategy for the computational tasks, i.e., offloading to AeBSs or executing them locally [39]. AeBSs perform local training, and the model parameters are uploaded to an HAP for federated model aggregation. We chose an HAP as a global node because of its powerful computing power and lower latency compared to satellites. In addition, there are almost no other obstacles in the air except the aircraft itself, which makes the communication link better and more reliable than the ground communication. AeBS n trains its own two networks Q_1^n and Q_2^n together, in which Q_1^n is responsible for deployment and Q_2^n is responsible for the offloading policy.

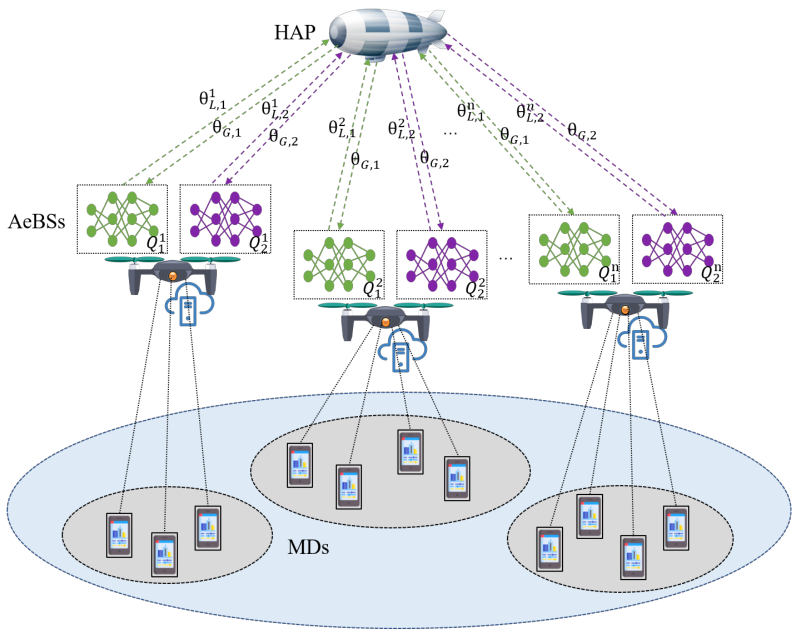

Figure 1. Federated deep-reinforcement-learning-based joint computation offloading and deployment control.

3.1. Communication Model

Since the line-of-sight (LoS) channel is dominant in air-to-ground links, we only considered the LoS propagation characteristics between AeBSs and MDs. Therefore, the channel gain between MD i and AeBS j can be obtained as:

$$h_{i,j} = g_0 d_{i,j}^{-2}, \tag{1}$$

where $d_{i,j}$ is the distance between MD i and AeBS j, which can be calculated by the locations of MD i and AeBS j. g_0 represents the channel gain at the reference distance of 1 m.

The data transmission rate between MD i and AeBS j is

$$R_{i,j} = B\log\left(1 + \frac{P_i h_{i,j}}{\sigma^2}\right), \tag{2}$$

where P_i denotes the transmit power of MD i and σ^2 represents the noise power.

3.2. Computation Model

For MD i with a computation task, D_i represents the size of the input data and S_i represents the CPU cycles required to process 1 bit of data. Let $A = \{a_{i,j}\}_{M\times(N+1)}$ represent the offloading decision of MDs. If the computation task in MD i is offloaded to AeBS j for computing, $a_{i,j} = 1$. If the computation task in MD i is computed locally, $a_{i,0} = 1$. Therefore, there are two kinds of computation modes that each MD can choose: local execution and offloading execution.

3.2.1. Local Execution

According to the definition of a task, the total CPU cycles required to execute the task is $S_i \times D_i$. Then, the time required to process a task in MD i is

$$T_i^{local} = \frac{S_i \times D_i}{f_i^{MD}}, \tag{3}$$

where f_i^{MD} is the computational capacity of MD i.

3.2.2. Offloading Execution

In this case, processing the task requires the following three steps. Firstly, MD i transmits the task data that need to be processed to AeBS j, which takes time $T_{i,j}^{tran}$. Secondly, AeBS j computes the data, which takes time T_j^{AeBS}. Thirdly, AeBS j transmits the result back to MD i, which takes a tiny amount of time $T_{j,i}^{back}$, and we usually neglect it. Thus, we have

$$T_{i,j}^{tran} = \frac{D_i}{R_{i,j}}, \tag{4}$$

$$T_j^{\text{AeBS}} = \frac{S_i \times D_i}{f_j^{\text{AeBS}}}, \tag{5}$$

where f_j^{AeBS} denotes the computational capacity of AeBS j. Therefore, the total time cost of offloading execution can be obtained as:

$$T_{i,j}^{off} = \frac{D_i}{R_{i,j}} + \frac{S_i \times D_i}{f_j^{\text{AeBS}}}. \tag{6}$$

3.3. Energy Model

For local execution, only the computation energy consumption needs to be considered as there is no data transmission. The energy consumption can be calculated as [40]:

$$E_i^{local} = \delta {f_i^{MD}}^2 S_i D_i, \tag{7}$$

where δ is an energy efficiency parameter, which is related to the chip architecture.

For offloading execution, the energy consumption includes transmission consumption and computation consumption, which can be written as:

$$E_{i,j}^{off} = P_i T_{i,j}^{tran} + \delta {f_j^{\text{AeBS}}}^2 S_i D_i. \tag{8}$$

3.4. Problem Formulation

In order to provide better service while taking into account the limited energy of AeBSs, it is necessary to minimize the system task processing delay and energy consumption. In this paper, we jointly optimized the AeBSs deployment and offloading strategy, aiming to minimize the weighted sum of task processing time and AeBS energy consumption, which can be written as:

$$\begin{aligned} \min_{X,Y,A} \, & w_1 \sum_{j=1}^{N} \sum_{i=1}^{M} \left(a_{i,0} T_i^{local} + a_{i,j} T_{i,j}^{off} \right) \\ & + w_2 \sum_{j=1}^{N} \sum_{i=1}^{M} \left(a_{i,0} E_i^{local} + a_{i,j} E_{i,j}^{off} \right), \\ & C1 : x_{min} < x_j < x_{max}, \\ & C2 : y_{min} < y_j < y_{max}, \\ & C3 : \sum_{j=0}^{N} a_{i,j} = 1, \\ & C4 : a_{i,j} \in \{0,1\}, \forall i, j, \end{aligned} \tag{9}$$

where $\forall i \in \{1, 2, \ldots, M\}$ and $\forall j \in \{1, 2, \ldots, N\}$. w_1 and w_2 denote the weights of the task processing time and energy consumption, respectively. Constraints $C1$ and $C2$ limit the range of movement of the AeBSs, and C3 and $C4$ indicate that each MD can either offload its task to an AeBS or execute the task locally and cannot partially offload its task. It is worth mentioning that, for such mixed-integer programming problems, with multi-objective optimization, it is difficult for traditional optimization algorithms to find the optimal solution in a short time, which is unacceptable for user computing tasks that change in real-time. To meet the real-time and complexity requirements, we propose FedDCO based on deep reinforcement learning, which can complete the deployment of AeBSs and obtain the offloading solution in a short time.

4. Federated Deep-Reinforcement-Learning-Based AeBS Deployment and Computation Offloading

The optimization problem in Equation (9) is defined as a mixed-integer programming problem, which is often difficult to find solutions to quickly. We designed an algorithm based on federated deep reinforcement learning called FedDCO, where each AeBS is equipped with a Q_1^n network responsible for generating deployment schemes and a Q_2^n network responsible for generating offloading policies. For an AeBS, it is unnecessary to focus on the MDs with weak channel gain, which will lead to information redundancy. Each AeBS only needs to pay attention to the information of the MDs that are within a certain distance. Therefore, we used the K-nearest-neighbor algorithm to divide the association between AeBSs and MDs. Then, each AeBS first moves to the optimal location using the Q_1^n network, followed by using the Q_2^n network to generate the offloading policies for nearby MDs. The basic components of FedDCO are as follows.

State: For Q_1^n of AeBS n, the state $s_1^n(t)$ includes the computational capacity of AeBS n and its associated MDs, the amount of computational tasks for its associated MDs, and the channel gain between AeBS n and MDs. The state $s_1(t)$ is the collection of each AeBS's state $s_1^n(t)$.

For Q_2^n of AeBS n, the state $s_2^n(t)$ also consists of the computational capacity of AeBS n and its associated MDs, the amount of computational tasks for its associated MDs, the channel gain between AeBS n and MDs, the number of MDs, which still have not generated a computation offloading decision, and a vector indicating the computation offloading policy for each MD. The state $s_2(t)$ is the collection of each AeBS state $s_2^n(t)$.

Action: For Q_1^n, the action $a_1^n(t)$ is the movement of AeBS n, including moving backward or forward, left or right, or remaining stationary. For Q_2^n, the action is the offloading policy of one MD, i.e., local execution or offloading execution.

Reward: We combine the impact of task latency and energy consumption in the overall task and define the reward function as $r_c - w_1 \sum_{j=1}^{N} \sum_{i=1}^{M} \left(a_{i,0} T_i^{local} + a_{i,j} T_{i,j}^{off} \right)$ $- w_2 \sum_{j=1}^{N} \sum_{i=1}^{M} \left(a_{i,0} E_i^{local} + a_{i,j} E_{i,j}^{off} \right)$, where r_c is a nonnegative constant.

We let $\theta_{L,1}^n$ represent the weight of Q_1^n, which is responsible for generating deployment decisions. $\theta_{L,2}^n$ represents the weight of Q_2^n, which generates the offloading policies. The updating processes of the two networks follow the strategy and principle of the classical DQN. The optimal Q function for AeBS n can be defined as:

$$Q^*(s_n, a_n) = \max_{\pi} {}_n \mathbb{E}\Big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s_n, a_t = a_n, \pi_n \Big], \quad (10)$$

which denotes the maximum expectation of the sum of the future reward at each time step t that can be obtained by the strategy $\pi_n = P(a_n \mid s_n)$ after observing state s_n and taking action a_n and γ is the discount factor. The DQN algorithm adopts the neural network to parameterize the approximation function $Q(s_n, a_n; \theta_n)$, where $\theta_{L,\zeta}^n$ is the weight parameter of the local neural network of AeBS n. The DQN utilizes the experience pool and stores the experiences of the agent at each time step t all in the dataset. During algorithm learning, the minibatches of the experiences are randomly sampled from the pool of stored samples to update the network. The loss function is calculated to measure the error between the prediction and the target value, which is:

$$\mathcal{L}_n\left(\theta^n_{L,\zeta}\right) = \mathbb{E}\left\{\left[y_n - Q\left(s_n, a_n; \theta^n_{L,\zeta}\right)\right]^2\right\}, \zeta \in \{1,2\}, \tag{11}$$

where $Q\left(s_n, a_n; \theta^n_{L,\zeta}\right)$ is the Q value in the Q-network with the current state s_n as the input; y_n is the target value and can be calculated as $y_n = r + \gamma \cdot \max_{a'_n} Q\left(s_n, a'_n; \theta^n_{L,\zeta}{}'\right)$. The Q-network is optimized in each iteration to minimize the loss function. We adopted the stochastic gradient descent method to optimize the loss function and update the weight parameters of the Q-networks. The DQN algorithm introduces two neural networks, the Q-network and the target Q-network, which have the same structure, but different parameters, and the parameters of the target Q-network are periodically updated according to the parameters in the Q-network.

After each AeBS trains its two networks based on its state, it uploads the parameters of Q_1 and Q_2 to the HAP for model aggregation at regular intervals f_a. The parameter f_a is set because it is unnecessary for AeBSs to upload model parameters to the HAP for each episode of training to save costs. The weight of the global model can be obtained as follows:

$$\theta_{G,\zeta} = \frac{1}{N}\sum_{n=1}^{N} \theta^n_{L,\zeta}, \zeta \in \{1,2\}, \tag{12}$$

where $\theta_{G,\zeta}$ is the weight of the global network and N is the number of AeBSs. The global network parameters are sent back to the AeBSs for updating their own local networks.

The detailed steps of the FedDCO algorithm are shown in Algorithm 1. We also depict the whole training process of the proposed FedDCO algorithm in Figure 2.

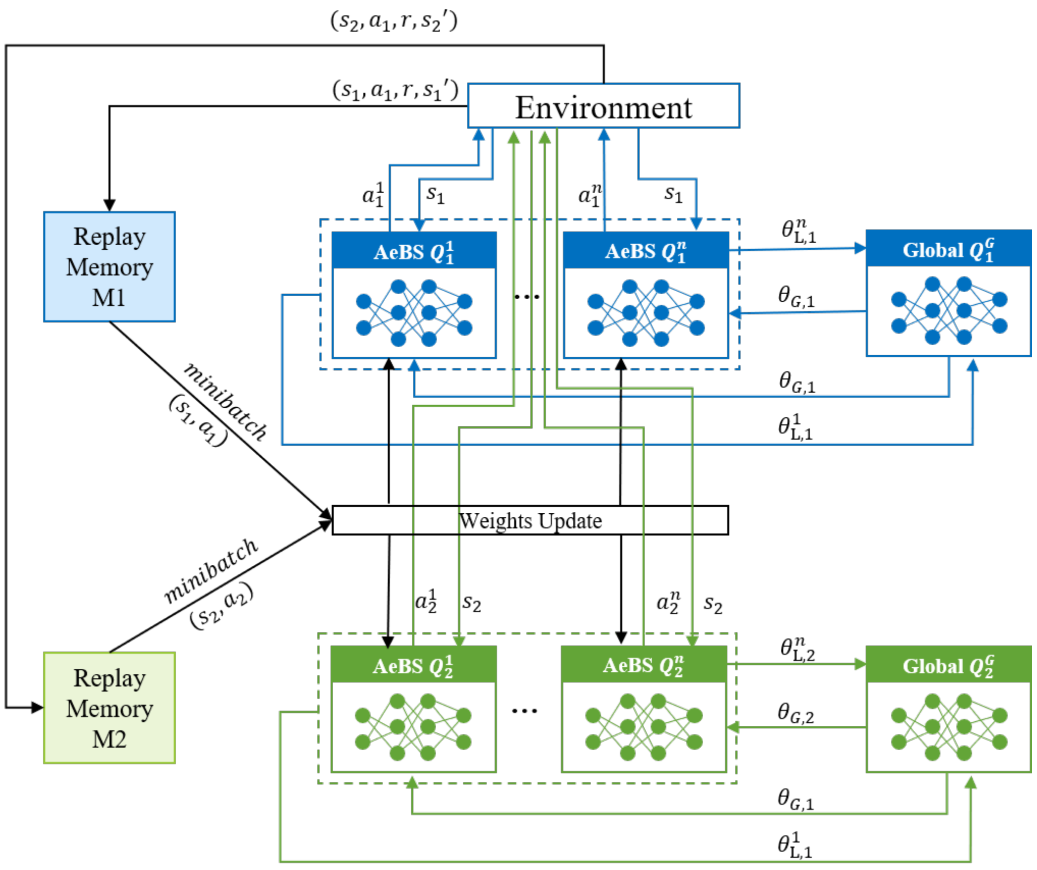

Figure 2. Flow chart of FedDCO.

Algorithm 1: Proposed FedDCO algorithm

```
1  Input: maximum number of episodes, states of AeBSs, locations of AeBSs and
     MDs
2  Output: deployment and offloading policy for each AeBS and MD
3  Initialize: replay memory size $M_1, M_2$, online network $Q_1^n, Q_2^n$ and target network
     $Q_1^{n\prime}, Q_2^{n\prime}$ for each AeBS $n$, the distribution of MDs, and the aggregation frequency
     of federated learning $f_a$
4  Initialize:
5  for $episode \leftarrow 1 : max\_episode$ do
6      Initialize the locations of AeBSs, and obtain the initial state $s_1$(t), $s_2$(t)
7      for $step \leftarrow 1 : max\_step$ do
8          Obtain the AeBS-MD association via the K-nearest-neighbor algorithm
9          for $AeBS \leftarrow 1 : N$ do
10             Choose an action $a_1^n(t)$ according to the $\varepsilon$-greedy policy: with
                 probability $1 - \varepsilon$; AeBS $n$ selects action
                 $a_1^n(t) = argmaxQ_1^n(s_1^n(t), a_1^n(t); \theta_{L,1}^n)$, otherwise it selects a random
                 action
11             Execute action $a_1^n(t)$
12         for $MD \leftarrow 1 : M$ do
13             Choose an action $a_2^n(t)$ according to the $\varepsilon$-greedy policy: with
                 probability $1 - \varepsilon$; its associated AeBS selects action
                 $a_2^n(t) = argmaxQ_2^n(s_2^n(t), a_2^n(t); \theta_{L,2}^n)$, otherwise it selects a random
                 action
14             Execute action $a_2^n(t)$
15             Observe reward $r$, and obtain the new state
16             Store $(s_2(t), a_2(t), r(t), s_2(t+1))$ in replay memory $M_2$
17         Store $(s_1(t), a_1(t), r^*(t), s_1(t+1))$ in replay memory $M_1$ ($r^*$ is the best
             reward in the current episode)
18         for $AeBS \leftarrow 1 : N$ do
19             Perform a gradient descent step on the loss function according to
                 Equation (11)
20             Every C steps, update the target network for each AeBS $n$
21     if $episode \ \ mod \ \ f_a$=0 then
22         Each AeBS uploads its $\theta_{L,1}^n$ and $\theta_{L,2}^n$ weights to the global node for model
             aggregation according to Equation (12), respectively.
23         The global node sends the aggregated global model weight of $\theta_{G,1}$ and $\theta_{G,2}$
             back to the AeBSs, and each AeBS updates its own model.
```

5. Simulation Results and Discussions

We selected a 1 km × 1 km area for simulation, and the performance metrics of the simulation were the total task processing time and energy consumption. The main simulation parameters are given in Table 2, which refer to [41]. The Q1- and Q2-networks of each AeBS were three-layer fully connected neural networks and used ReLU as the activation function. The simulation environment was Python 3.7 and Pytorch 1.9.1 and was run on a computer with i5-12500H processor from Intel and an RTX3060 GPU from Nvidia.

Table 2. Main simulation parameters.

Simulation Parameters	Values
AeBS altitude H	100 m
Transmit power P_i	0.5 W
Channel bandwidth B	1 MHz
Reference channel gain g_0	10,096
Energy efficiency parameter δ	2
Noise σ^2	5×10^{-5} W
ϵ in ϵ-greedy	0.1
Memory size	10,000
Batch size	512
Discount factor	0.97

We compared our FedDCO scheme with three other approaches named MADCO, K-means-based, and throughput-first, to observe the performance gain brought by federated learning, simultaneous training of two deep Q-networks, and location adjustment of the AeBSs, respectively [41]. The description of these schemes is listed as follows:

1. FedDCO: Each AeBS has two deep Q-networks, Q_1 and Q_2, and they are trained simultaneously to generate deployment and offloading policies. During the training process, AeBSs upload the Q_1 and Q_2 weights instead of the raw state and action data to the global node for model aggregation, and the global node sends the aggregated global model weight of Q_1 and Q_2 back to the AeBSs, then each AeBS updates its own model.
2. MADCO: MADCO optimizes the AeBS deployment schemes and offloading strategies by training two neural networks together to minimize the latency and energy consumption of computational task processing. Each AeBS exchanges action and state information with each other when making decisions. Its settings for the input/output, parameters, and DNN structure are consistent with FedDCO.
3. K-means: AeBSs are deployed based on the MD distribution through the K-means algorithm. The number of clusters of K-means was set as the number of AeBSs, and then, each AeBS is deployed directly above each cluster center of MDs. Specifically, the maximum number of iterations of K-means was 300, and if the sum of squares within all clusters between two iterations is less than 1×10^{-4}, the iteration is terminated. After the location of AeBSs is fixed, the offloading policy is generated through the Q_2-network, whose input/output settings, parameter settings, and network structure are the same as Q_2 in FedDCO.
4. Throughput-first: AeBSs are first deployed based on the Q_1-network with the goal of maximizing throughput, and the offloading policy is later generated through the Q_2-network. The settings of the input/output, parameters. and DNN structure in throughput-first are also consistent with FedDCO.

Figure 3 describes the convergence process of the above four algorithms, in the case that the number of AeBSs is 8 and the number of MDs is 60. We normalized the reward to reduce the magnitude difference between the task processing delay and energy consumption. The average reward of the proposed FedDCO increased rapidly in the first 100 episodes. After 150 episodes, the algorithm converged, and the reward became stable. It can be found that the convergence speed of FedDCO was faster than that of MADCO, which indicates that federated learning can improve the convergence speed of the model. K-means uses a clustering algorithm to deploy AeBS positions; although its training time was fast, its performance was the lowest. This is because it does not take advantage of the mobility and self-adjustment of the deployment location of AeBSs to obtain better performance.

In Figure 4, we mainly evaluate the effect of different algorithms with different numbers of MDs in the following three typical scenarios:

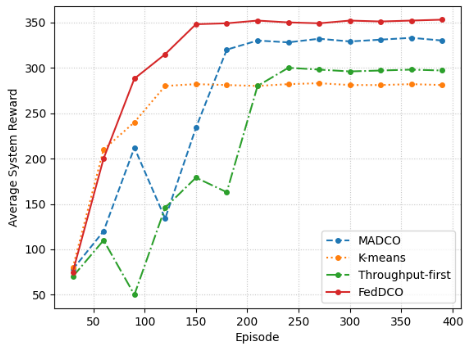

Figure 3. The reward curve of the four algorithms in the training process.

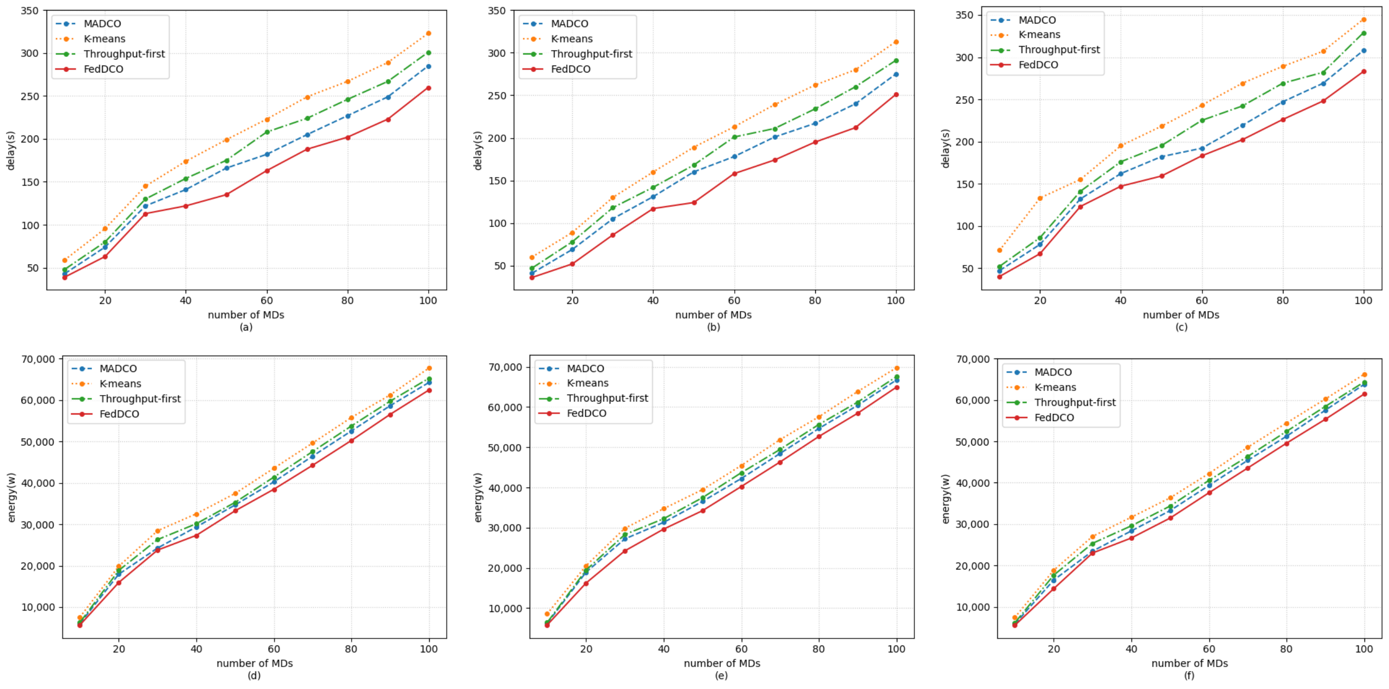

Figure 4. Comparison of network performance metrics at different MD quantities in different communication scenarios. (**a**–**c**) are the task processing delay in the general communication scenario, delay-sensitive scenario, and energy-sensitive scenario, respectively. (**d**–**f**) are the energy consumption in the general communication scenario, delay-sensitive scenario, and energy-sensitive scenario, respectively.

1. General communication scenario: In this scenario, we regarded delay and energy consumption as equally important indicators. Thus, we set $(w_1, w_2) = (1, 1)$ in the simulation.
2. Delay-sensitive scenario: There may be some real-time services in the network, which makes the MDs sensitive to delay. For this scenario, we set $(w_1, w_2) = (2, 1)$ in the simulation.
3. Energy-sensitive scenario: For some aerial platforms with limited payload capacity, such as small multi-rotor unmanned aerial vehicles, the battery capacity is limited, so it is necessary to reduce the energy consumption as much as possible to ensure the completion of the mission. For this scenario, we set $(w_1, w_2) = (1, 2)$ in the simulation.

Figures 4a–c depict the task processing delay for the three different scenarios, with different numbers of MDs ranging from 10 to 100 [42]. By comparing different algorithms, it can be observed that our proposed FedDCO obtained the lowest task processing latency in all of three cases. It can be seen that K-means had a longer task processing time in most cases, which may be due to the fact that it does not use a neural network to obtain a better deployment position. Figure 4d–f show the energy consumption under each scenario. FedDCO still achieved the best results compared to the other three algorithms. K-means and throughput-first performed similarly, probably because Q_2 is effective in minimizing energy consumption, leading them to make similar offloading strategies. By comparing the network performance metrics in different scenarios, it can be discovered that the delay of each scheme in the delay-sensitive scenario was basically smaller than that in the general communication scenario, but at the cost of more energy consumption. In the energy-constrained scenario, the agent's strategy prefers a lower energy consumption, and the task processing latency in this scenario increases slightly. To sum up the above figures, with the increase of the MDs' number, the key indicators, task processing delay, and energy consumption of the system were all on the rise. Taking the three indicators into consideration, the proposed FedDCO achieved the optimal performance in various communication scenarios compared to the other algorithms. The K-means-based scheme performed worst among these algorithms since it does not take advantage of adjusting the position of the AeBSs. This indicates that the deployment design of the AeBSs is a very important factor to be considered in the aerial edge computing network.

Figure 5 shows the performance of the four algorithms described in this article at different data sizes of the tasks. It can be noticed that, with the increasing data size of computing tasks, task processing delay and energy consumption both showed an upward trend. This was attributed to the fact that there were more data to be transmitted and processed, which brought a burden to the system and led to the degradation of the system performance. In the case of different data sizes of the tasks, our proposed FedDCO had a lower delay and energy consumption than the other three algorithms.

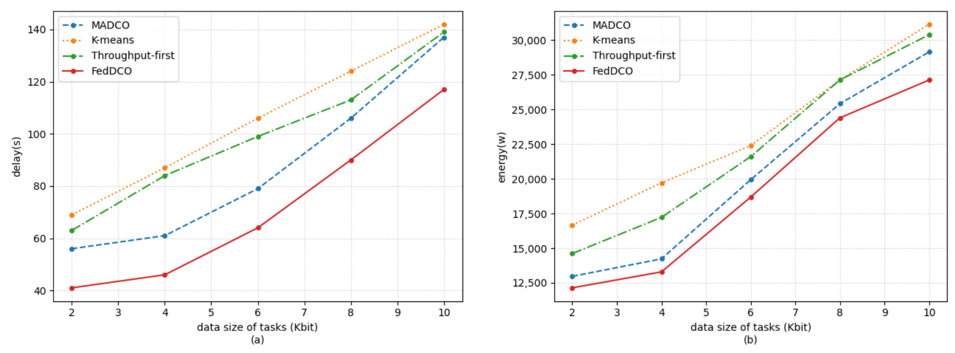

Figure 5. Comparison of network performance metrics at different data sizes of the tasks. (**a**) Task processing delay. (**b**) Energy consumption.

Figure 6 illustrates the performance of the four algorithms described in this article at different computational capacities of the AeBSs. With the enhancement of the computing capability of the AeBSs, the delay of the system decreased, but the system consumed more energy. This is because, when the computational power of the AeBSs increased, the tasks on the MDs tended to be offloaded to the AeBSs, which resulted in an increase in both the signal transmission energy consumption and AeBS computation energy consumption. By comparing the four algorithms, it can be found that our proposed algorithm FedDCO had the best performance, while the traditional machine learning algorithm K-means had a relatively poor performance.

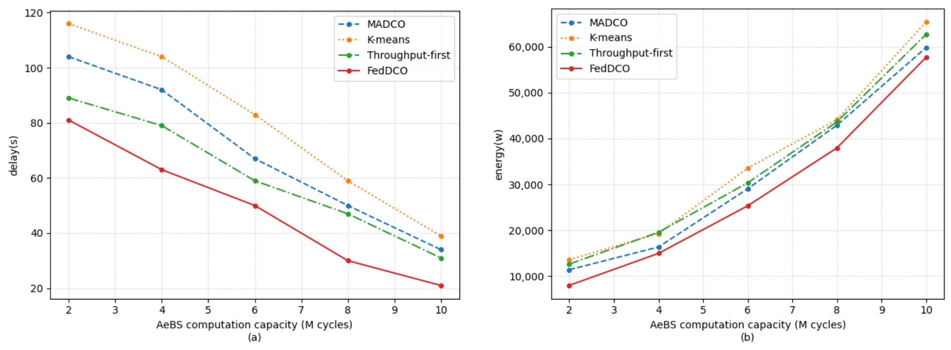

Figure 6. Comparison of network performance metrics at different computational capacities of the AeBSs. (**a**) Task processing delay. (**b**) Energy consumption.

Figure 7 shows the training time of FedDCO, MADCO, K-means, and throughput-first with different numbers of AeBSs. As the number of AeBSs in the system increased, the problem complexity increased, so the training time of FedDCO and MADCO also increased. The training time of FedDCO was smaller than that of MADCO and throughput-first, which indicates that federated learning can accelerate convergence and improve algorithm efficiency. This is owed to the fact that federated learning can solve the problem of the difficult convergence of agents by exchanging experience through parameter aggregation. In the K-means-based scheme, the K-means algorithm was adopted to decide the deployment position of the AeBSs instead of training a DRL model, which made its training speed relatively fast. Especially when the number of AeBSs increased, this gap became larger. However, the performance of the K-means-based scheme was not as good as FedDCO.

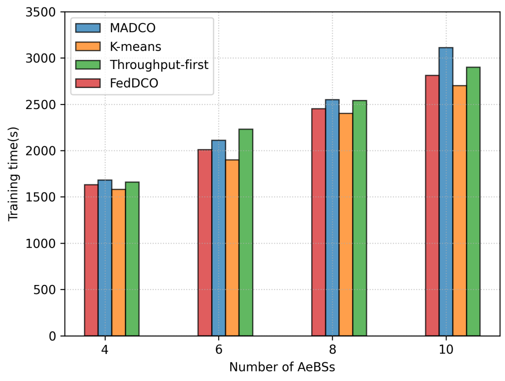

Figure 7. Training time of different algorithms.

Generally speaking, the simulation results above proved that the proposed FedDCO outperformed the other algorithms in different communication scenarios. Moreover, FedDCO also had the advantages of fast convergence and a short training time, which is very suitable for the dynamic network environment in aerial edge computing networks.

6. Conclusions

In this paper, we proposed an approach called FedDCO to address the joint optimization problem of AeBSs' deployment and computation offloading in an aerial edge computing network with the goal of minimizing the task processing time and energy consumption. We designed a training mechanism based on federated deep reinforcement learning, where low-altitude AeBSs train their local neural networks individually and an HAP plays the role of a global node for model aggregation. The simulation results showed that our proposed approach can achieve better performance in various communication scenarios compared with other benchmark schemes.

Author Contributions: Conceptualization, L.L. and F.Q.; funding acquisition, L.L. and W.X.; investigation, Y.Z. and W.X.; methodology, F.Q. and F.Z.; supervision, L.L.; validation, H.H. and H.Z.; visualization, H.H. and H.Z.; writing—original draft, Y.Z. and H.H.; writing—review and editing, F.Q. and F.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (No. 2020YFB1806700).

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

Abbreviations

The following abbreviations are used in this manuscript:

MEC	Mobile edge computing
HAP	High altitude platform
AeBS	Aerial base station
MD	Mobile device
FL	Federated learning
RL	Reinforcement learning
DRL	Deep reinforcement learning
DQN	Deep Q-network
DDPG	Deep deterministic policy gradient
MADRL	Multi-agent deep reinforcement learning
LOS	Line-of-sight

References

1. Ji, B.; Wang, Y.; Song, K.; Li, C.; Wen, H.; Menon, V.G.; Mumtaz, S. A Survey of Computational Intelligence for 6G: Key Technologies, Applications and Trends. *IEEE Trans. Ind. Informatics* **2021**, *17*, 7145–7154. [CrossRef]
2. Cui, H.; Zhang, J.; Geng, Y.; Xiao, Z.; Sun, T.; Zhang, N.; Liu, J.; Wu, Q.; Cao, X. Space-air-ground integrated network (SAGIN) for 6G: Requirements, architecture and challenges. *China Commun.* **2022**, *19*, 90–108. [CrossRef]
3. Pham, Q.V.; Ruby, R.; Fang, F.; Nguyen, D.C.; Yang, Z.; Le, M.; Ding, Z.; Hwang, W.J. Aerial Computing: A New Computing Paradigm, Applications, and Challenges. *IEEE Internet Things J.* **2022**, *9*, 8339–8363. [CrossRef]
4. Xu, Y.; Zhang, T.; Liu, Y.; Yang, D.; Xiao, L.; Tao, M. UAV-Assisted MEC Networks With Aerial and Ground Cooperation. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7712–7727. [CrossRef]
5. Shakeri, R.; Al-Garadi, M.A.; Badawy, A.; Mohamed, A.; Khattab, T.; Al-Ali, A.K.; Harras, K.A.; Guizani, M. Design Challenges of Multi-UAV Systems in Cyber-Physical Applications: A Comprehensive Survey and Future Directions. *IEEE Commun. Surv. Tutorials* **2019**, *21*, 3340–3385. [CrossRef]
6. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
7. Zheng, Q.; Yang, M.; Yang, J.; Zhang, Q.; Zhang, X. Improvement of Generalization Ability of Deep CNN via Implicit Regularization in Two-Stage Training Process. *IEEE Access* **2018**, *6*, 15844–15869. [CrossRef]
8. You, L.; Jiang, H.; Hu, J.; Chang, C.H.; Chen, L.; Cui, X.; Zhao, M. GPU-accelerated Faster Mean Shift with euclidean distance metrics. In Proceedings of the 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), Alamitos, CA, USA, 27 June–1 July 2022; pp. 211–216. [CrossRef]
9. Zhao, M.; Chang, C.H.; Xie, W.; Xie, Z.; Hu, J. Cloud Shape Classification System Based on Multi-Channel CNN and Improved FDM. *IEEE Access* **2020**, *8*, 44111–44124. [CrossRef]
10. Jin, B.; Cruz, L.; Gonçalves, N. Pseudo RGB-D Face Recognition. *IEEE Sens. J.* **2022** . [CrossRef]
11. Arshad, M.A.; Khan, S.H.; Qamar, S.; Khan, M.W.; Murtza, I.; Gwak, J.; Khan, A. Drone Navigation Using Region and Edge Exploitation-Based Deep CNN. *IEEE Access* **2022**, *10*, 95441–95450. [CrossRef]
12. Liang, F.; Yu, W.; Liu, X.; Griffith, D.; Golmie, N. Toward Edge-Based Deep Learning in Industrial Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 4329–4341. [CrossRef]
13. Zhao, Y.; Zhou, F.; Li, W.; Gao, Y.; Feng, L.; Yu, P. 3D Deployment and User Association of CoMP-assisted Multiple Aerial Base Stations for Wireless Network Capacity Enhancement. In Proceedings of the 2021 17th International Conference on Network and Service Management (CNSM), Izmir, Turkey, 25–29 October 2021; pp. 56–62. [CrossRef]
14. Zhang, Y.; Mou, Z.; Gao, F.; Jiang, J.; Ding, R.; Han, Z. UAV-Enabled Secure Communications by Multi-Agent Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2020**, *69*, 11599–11611. [CrossRef]
15. Xu, Y.; Bhuiyan, M.Z.A.; Wang, T.; Zhou, X.; Singh, A. C-fDRL: Context-Aware Privacy-Preserving Offloading through Federated Deep Reinforcement Learning in Cloud-Enabled IoT. *IEEE Trans. Ind. Inform.* **2022**. [CrossRef]
16. Yang, W.; Xiang, W.; Yang, Y.; Cheng, P. Optimizing Federated Learning With Deep Reinforcement Learning for Digital Twin Empowered Industrial IoT. *IEEE Trans. Ind. Inform.* **2022**. [CrossRef]

17. Tehrani, P.; Restuccia, F.; Levorato, M. Federated Deep Reinforcement Learning for the Distributed Control of NextG Wireless Networks. In Proceedings of the 2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN), Angeles, CA, USA, 13–15 December 2021; pp. 248–253. [CrossRef]
18. Zhang, X.; Peng, M.; Yan, S.; Sun, Y. Deep-Reinforcement-Learning-Based Mode Selection and Resource Allocation for Cellular V2X Communications. *IEEE Internet Things J.* **2020**, *7*, 6380–6391. [CrossRef]
19. Jia, Z.; Wu, Q.; Dong, C.; Yuen, C.; Han, Z. Hierarchical Aerial Computing for Internet of Things via Cooperation of HAPs and UAVs. *IEEE Internet Things J.* **2022**. [CrossRef]
20. 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]
21. Truong, T.P.; Tran, A.T.; Nguyen, T.M.T.; Nguyen, T.V.; Masood, A.; Cho, S. MEC-Enhanced Aerial Serving Networks via HAP: A Deep Reinforcement Learning Approach. In Proceedings of the 2022 International Conference on Information Networking (ICOIN), Jeju-si, Korea, 12–15 January 2022; pp. 319–323. [CrossRef]
22. Zhou, F.; Wu, Y.; Sun, H.; Chu, Z. UAV-Enabled Mobile Edge Computing: Offloading Optimization and Trajectory Design. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [CrossRef]
23. Huang, L.; Feng, X.; Feng, A.; Huang, Y.; Qian, L.P. Distributed Deep Learning-based Offloading for Mobile Edge Computing Networks. *Mob. Networks Appl.* **2022**, *27*, 1123–1130. [CrossRef]
24. Yang, B.; Cao, X.; Yuen, C.; Qian, L. Offloading Optimization in Edge Computing for Deep-Learning-Enabled Target Tracking by Internet of UAVs. *IEEE Internet Things J.* **2021**, *8*, 9878–9893. [CrossRef]
25. Min, M.; Xiao, L.; Chen, Y.; Cheng, P.; Wu, D.; Zhuang, W. Learning-Based Computation Offloading for IoT Devices With Energy Harvesting. *IEEE Trans. Veh. Technol.* **2019**, *68*, 1930–1941. [CrossRef]
26. Chen, X.; Zhang, H.; Wu, C.; Mao, S.; Ji, Y.; Bennis, M. Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning. *IEEE Internet Things J.* **2019**, *6*, 4005–4018. [CrossRef]
27. 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]
28. Li, J.; Liu, Q.; Wu, P.; Shu, F.; Jin, S. Task Offloading for UAV-based Mobile Edge Computing via Deep Reinforcement Learning. In Proceedings of the 2018 IEEE/CIC International Conference on Communications in China (ICCC), Beijing, China, 16–18 August 2018; pp. 798–802. [CrossRef]
29. Wang, L.; Huang, P.; Wang, K.; Zhang, G.; Zhang, L.; Aslam, N.; Yang, K. RL-Based User Association and Resource Allocation for Multi-UAV enabled MEC. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 741–746. [CrossRef]
30. Wang, X.; Ning, Z.; Guo, S. Multi-Agent Imitation Learning for Pervasive Edge Computing: A Decentralized Computation Offloading Algorithm. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 411–425. [CrossRef]
31. Wang, Z.; Rong, H.; Jiang, H.; Xiao, Z.; Zeng, F. A Load-Balanced and Energy-Efficient Navigation Scheme for UAV-Mounted Mobile Edge Computing. *IEEE Trans. Netw. Sci. Eng.* **2022**, *9*, 3659–3674. [CrossRef]
32. Wang, L.; Wang, K.; Pan, C.; Xu, W.; Aslam, N.; Hanzo, L. Multi-Agent Deep Reinforcement Learning-Based Trajectory Planning for Multi-UAV Assisted Mobile Edge Computing. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 73–84. [CrossRef]
33. Waqar, N.; Hassan, S.A.; Mahmood, A.; Dev, K.; Do, D.T.; Gidlund, M. Computation Offloading and Resource Allocation in MEC-Enabled Integrated Aerial-Terrestrial Vehicular Networks: A Reinforcement Learning Approach. *IEEE Trans. Intell. Transp. Syst.* **2022**. [CrossRef]
34. Nie, Y.; Zhao, J.; Gao, F.; Yu, F.R. Semi-Distributed Resource Management in UAV-Aided MEC Systems: A Multi-Agent Federated Reinforcement Learning Approach. *IEEE Trans. Veh. Technol.* **2021**, *70*, 13162–13173. [CrossRef]
35. Zhu, Z.; Wan, S.; Fan, P.; Letaief, K.B. An Edge Federated MARL Approach for Timeliness Maintenance in MEC Collaboration. In Proceedings of the 2021 IEEE International Conference on Communications Workshops (ICC Workshops), Montreal, QC, Canada, 14–23 June 2021; pp. 1–6. [CrossRef]
36. Zhu, Z.; Wan, S.; Fan, P.; Letaief, K.B. Federated Multiagent Actor–Critic Learning for Age Sensitive Mobile-Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 1053–1067. [CrossRef]
37. Wang, X.; Wang, C.; Li, X.; Leung, V.; Taleb, T. Federated Deep Reinforcement Learning for Internet of Things with Decentralized Cooperative Edge Caching. *IEEE Internet Things J.* **2020**, *7*, 9441–9455. [CrossRef]
38. Majidi, F.; Khayyambashi, M.R.; Barekatain, B. HFDRL: An Intelligent Dynamic Cooperate Cashing Method Based on Hierarchical Federated Deep Reinforcement Learning in Edge-Enabled IoT. *IEEE Internet Things J.* **2022**, *9*, 1402–1413. [CrossRef]
39. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutorials* **2017**, *19*, 2322–2358. [CrossRef]
40. 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.* **2020**, *19*, 2581–2593. [CrossRef]

41. Xu, Z.; Zhou, F.; Feng, L.; Wu, Z. MARL-Based Joint Computation Offloading and Aerial Base Stations Deployment in MEC Systems. In Proceedings of the 2022 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Bilbao, Spain, 15–17 June 2022; pp. 1–6. [CrossRef]
42. Dai, B.; Niu, J.; Ren, T.; Hu, Z.; Atiquzzaman, M. Towards Energy-Efficient Scheduling of UAV and Base Station Hybrid Enabled Mobile Edge Computing. *IEEE Trans. Veh. Technol.* **2022**, *71*, 915–930. [CrossRef]