electronics

Article

DDPG-Based Computation Offloading Strategy for Maritime UAV

Ziyue Zhao, Yanli Xu * and Qianlian Yu

Department of Electronics Engineering, College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China; 202210320015@stu.shmtu.edu.cn (Z.Z.); 202130310011@stu.shmtu.edu.cn (Q.Y.)
* Correspondence: ylxu@shmtu.edu.cn

Abstract

With the development of the maritime Internet of Things (MIoT), a large number of sensors are deployed, generating massive amounts of data. However, due to the limited data processing capabilities of the sensors and the constrained service capacity of maritime communication networks, the local and cloud data processing of MIoT are restricted. Thus, there is a pressing demand for efficient edge-based data processing solutions. In this paper, we investigate unmanned aerial vehicle (UAV)-assisted maritime edge computing networks. Under energy constraints of both UAV and MIoT devices, we propose a Deep Deterministic Policy Gradient (DDPG)-based maritime computation offloading and resource allocation algorithm to efficiently process MIoT tasks current form of UAV. The algorithm jointly optimizes task offloading ratios, UAV trajectory planning, and edge computing resource allocation to minimize total system task latency while satisfying energy consumption constraints. Simulation results validate its effectiveness and robustness in highly dynamic maritime environments.

Keywords: Deep Deterministic Policy Gradient (DDPG); computation offloading; Mobile Edge Computing (MEC); Unmanned Aerial Vehicle (UAV); maritime

check for updates

Academic Editor: Gianfranco Avitabile

Received: 14 July 2025
Revised: 13 August 2025
Accepted: 25 August 2025
Published: 25 August 2025

Citation: Zhao, Z.; Xu, Y.; Yu, Q. DDPG-Based Computation Offloading Strategy for Maritime UAV. *Electronics* **2025**, *14*, 3376. https://doi.org/10.3390/electronics14173376

Correction Statement: This article has been republished with a minor change. The change does not affect the scientific content of the article and further details are available within the backmatter of the website version of this article.

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Currently, the combination of Mobile Edge Computing (MEC) and Mobile Internet of Things has garnered widespread attention. The Internet of Things allows a wide range of smart devices, from simple smart home devices to complex industrial machinery, to gather extensive data and communicate with each other and with other internet-connected devices [1]. This seamless data exchange enables these devices to carry out a variety of tasks autonomously [2]. The proven success of terrestrial IoT systems has inspired significant research interest in developing maritime counterparts—the Maritime Internet of Things. Due to the high cost of infrastructure deployment, many maritime Internet of Things (MIoT) nodes lack sufficient computational power to process the raw data they collect. Therefore, data must be transferred to devices with richer computational resources for processing. On the one hand, computing resource constraints impede the implementation of the maritime Internet of Things, the issue of wireless network power consumption cannot be ignored. The finite wireless spectrum necessitates advanced channel coding and modulation techniques to mitigate interference. This typically increases power consumption, affecting not only transceivers but the entire radio access network [3]. Given that wireless networks are projected to significantly increase their global carbon footprint, potentially doubling within the next decade [4], a critical challenge in MIoT deployment is minimizing its conventional

energy usage to reduce both carbon emissions and environmental harm. Additionally, the vast number of MIoT devices deployed in challenging environments—such as underwater sensors and sea-based smart buoys—makes battery replacement or recharging logistically difficult and cost-ineffective [5]. Consequently, the efficient utilization of limited spectrum resources, minimization of unmanned aerial vehicle (UAV) energy consumption, and fulfillment of user Quality of Service (QoS) requirements have emerged as critical priorities demanding immediate resolution [6]. The maritime environment presents fundamentally different and more dynamic challenges for wireless communication and UAV operation compared to traditional terrestrial or urban settings. Unlike relatively static urban infrastructure, the ocean surface exhibits significant mobility due to waves and swell, causing time-varying signal reflection and scattering paths. Prevalent salt spray and fog introduce variable attenuation and absorption of radio waves [7]. Furthermore, rapidly changing weather conditions, including wind speed fluctuations and precipitation, not only impact the wireless channel quality (inducing higher latency fluctuations and packet loss risk) but also significantly affect UAV propulsion energy consumption due to increased wind resistance. This inherent spatio-temporal environmental dynamism results in substantially higher levels of uncertainty and variability in both communication link performance and UAV energy expenditure than typically encountered in city-based deployments. Consequently, designing robust computation offloading and resource allocation strategies capable of adapting to these dynamic conditions is paramount for effective maritime edge computing. The present paper focuses on a single UAV-assisted maritime edge computing network, proposing a Deep Deterministic Policy Gradient (DDPG)-based maritime computation offloading and resource allocation algorithm under the constraints of UAV and MIoT device energy consumption, with the objective of minimizing the total latency of maritime IoT devices. The utilisation of UAVs in communication networks offers distinct advantages, including rapid deployment, flexible mobility, and line-of-sight communications [8]. Existing UAV hardware with MEC server capabilities provides a feasible foundation for the proposed approach. For instance, commercial UAVs such as DJI Matrice 300 RTK can be equipped with edge computing modules like NVIDIA Jetson series, which offer 1.3 TFLOPS of computing power, sufficient to handle medium-scale tasks offloaded by maritime IoT devices. Additionally, modified versions of the U.S. military's RQ-4 Global Hawk UAV have achieved on-board edge computing functionality for real-time processing of maritime monitoring data, verifying the hardware feasibility of integrating MEC in UAVs for maritime scenarios. UAV-assisted maritime MEC networks are capable of meeting the computational demands of user devices, while concurrently reducing energy consumption and task latency [9].

In a word, we propose a novel DDPG-based maritime computation offloading and resource allocation algorithm for UAV-assisted edge computing networks. This algorithm uniquely jointly optimizes the task offloading ratios, UAV trajectory planning, and edge computing resource allocation, aiming to minimize the total system task latency under the stringent energy consumption constraints of both the UAV and maritime IoT devices. Extensive simulations validate the effectiveness and robustness of our approach in highly dynamic maritime environments.

Afterwards, Section 2 reviews related work on computation offloading, particularly in maritime and UAV-assisted contexts. Section 3 details the system model, including the communication and computation models, and formally states the optimization problem. Section 4 presents our proposed DDPG-based strategy, describing the Markov Decision Process formulation and the algorithm framework. Section 5 provides simulation results and performance analysis. Finally, Section 6 concludes the paper and discusses potential future work.

2. Related Works

Beyond serving as a repository of abundant biological and mineral resources, the ocean critically enables biodiversity research, global climate change investigation, maritime resource development, and maritime transportation systems [10,11]. In recent years, due to the rapid development of maritime activities, the ocean economy, and the Internet of Things [12], the Maritime Internet of Things has become an excellent solution for achieving ubiquitous connectivity in the maritime domain [13]. However, objective environmental factors, such as the cost of infrastructure deployment, hinder the MIoT's ability to process the collected raw data. This paper focuses on solving the problem of minimizing the total latency of maritime IoT devices under the energy constraints of both the UAV and the MIoT devices. Extensive research has already been conducted by scholars across various domains on computation offloading.

In the domain of connected vehicles, researchers have harnessed the combined advantages of Software Defined Network (SDN) and MEC to tackle issues related to high-definition (HD) map-based driving assistance and navigation safety enhancement [14]. For instance, Zhang et al. [15] introduced a mobile-aware hierarchical MEC framework to improve the offloading performance of resource-constrained smart devices. However, most existing task offloading studies in this field fail to adequately consider the load balancing of computational resources across edge servers, limiting the overall efficiency of the systems.

Deep Reinforcement Learning (DRL)-based methods have been widely explored for task offloading in various edge computing scenarios. Nevertheless, these approaches often struggle to adapt to new environments due to inefficient sampling mechanisms and the necessity for comprehensive retraining. To overcome these limitations, Wang et al. [16] employed a meta-reinforcement learning approach for task offloading, which significantly reduces gradient updates and sample requirements, enabling rapid adaptation to new environments. Despite these advancements, applying DRL to solve task offloading problems in complex and dynamic maritime networks remains a relatively untapped area.

In the context of next-generation maritime information systems, several task offloading algorithms for maritime MEC have been proposed, which meet the requirements for low-latency and high-reliability application services to some extent [17]. For example, some studies have analyzed the trade-off between communication delay and energy consumption in maritime networks and proposed joint optimization algorithms [18]. Others have developed task offloading models for nearshore and farshore scenarios, solved using optimization algorithms such as genetic algorithms and particle swarm optimization. However, heuristic algorithms typically require numerous iterations during the optimization process, and their computational performance deteriorates significantly in complex offloading environments, with no guarantee of solution quality. Discretizing continuous variables using Deep Q-Network (DQN) or Double Deep Q-Network (DDQN) disrupts the spatial continuity, making it difficult to identify the optimal policy, especially in scenarios where continuous actions, such as the mobility of Unmanned Aerial Vehicles, need to be optimized. Paper [19] proposed a hybrid deep learning-based offloading algorithm for a land-UAV MEC platform to minimize IoT device energy consumption. However, its limitation to processing only one device's input at a time renders it impractical. In contrast, [20] developed a multiagent DRL framework for joint V2V/V2I offloading to meet dual-link delay requirements, but suffers from slow convergence.

The present paper is concerned with the subject of UAV-assisted maritime edge computing networks. The objective is to minimize the total latency of maritime IoT devices by means of joint optimisation of task offloading ratios, UAV mobility, and resource allocation. The problem is modelled as a Markov Decision Process and solved using the DDPG algorithm. This paper proposes a novel DDPG-based maritime computation offloading

and resource allocation strategy. Different from previous studies, our approach comprehensively considers the dynamic characteristics of the maritime environment and the continuous nature of UAV mobility, enabling more efficient and accurate decision-making. The simulation results demonstrate that our DDPG-based strategy outperforms baseline methods in terms of latency reduction and convergence speed, effectively optimizing UAV trajectory and resource allocation in uncertain environments. This paper not only fills the gap in the research of DDPG-based computation offloading strategies in maritime networks but also provides a valuable reference for future research on multi-UAV-assisted maritime IoT networks.

3. System Model

This section formally presents the system model for the UAV-assisted maritime MEC network. It details the key components: the communication model governing data transmission between MIoT devices and the UAV, the computation model describing local and edge task processing, and the formulation of the joint optimization problem aiming to minimize system latency under energy constraints.

3.1. Communication Model

This paper considers a UAV-assisted maritime mobile edge computing system, as illustrated in Figure 1, which comprises a UAV equipped with an MEC server and N MIoT devices, including but not limited to buoys, vessels, and various sensors, denoted as $\mathcal{N} = \{1, 2, \cdots, N\}$. Due to the limited computing capabilities of MIoT devices, they are required to offload part of their computational tasks to the UAV's MEC server. The UAV can provide communication and computational services to all MIoT devices but only serves one MIoT device at a time. The UAV can be used as an edge node to provide edge computing and caching functionalities to MIoT devices. After collecting sensing data from MIoT devices, the UAV processes the data through the edge server and returns the results. Additionally, the entire system operates within discrete time slots of equal length T. The system adopts a time division synchronization approach to coordinate the UAV and MIoT devices. The entire operation cycle is divided into discrete time slots $t = 1, 2, 3, ..., T$ of equal length. During each time slot, the UAV interacts (for communication and computation) with only one MIoT device sequentially: first completing the device's task offloading (data upload), then processing the task via the edge server, and finally returning the result (transmission time is negligible due to the small size of result data). In the next time slot, the UAV switches to the next device, and so on. This time division mechanism ensures no resource conflicts between devices, achieving system-level time synchronization.

Assume the position coordinates of MIoT device $i \in \{1, 2, \cdots, i, \cdots, N\}$ are denoted as $\mathbf{p}_i(t) = [x_i(t), y_i(t)]^T$. It is posited that, for the duration of each designated time period, the UAV maintains a constant altitude, denoted by H, during the flight phase, with the starting and ending points of the UAV being $\mathbf{q}(t) = [x(t), y(t)]^T$ and $\mathbf{q}(t+1) = [x(t+1), y(t+1)]^T$, respectively. Suppose the initial horizontal coordinates of the UAV are $q_0 = [x_0, y_0]^T$. In time slot t, the UAV moves from position $\mathbf{q}(t)$ to a new hovering position $\mathbf{q}(t+1)$ with speed $v(t) \in [0, v_{max}]$ and angle $\theta(t) \in [0, 2\pi]$. The coordinates of the new hovering position can be expressed as:

$$q(t+1) = \left[x(t) + v(t)t_{fly}\cos\theta(t), y + v(t)t_{fly}\sin\theta(t)\right]^T \tag{1}$$

It is assumed that the communication channel between MIoT device and UAV is mainly composed of Line of Sight (LoS). This assumption is based on empirical measurements

for near sea-surface communications at relevant frequencies [21]. The Euclidean distance between UAV and MIoT device i can be written as:

$$d_i(t) = \sqrt{[x(t+1) - x_i(t)]^2 + [y(t+1) - y_i(t)]^2 + H^2} = \sqrt{\|\mathbf{q}(t+1) - \mathbf{p}_i(t)\|^2 + H^2} \quad (2)$$

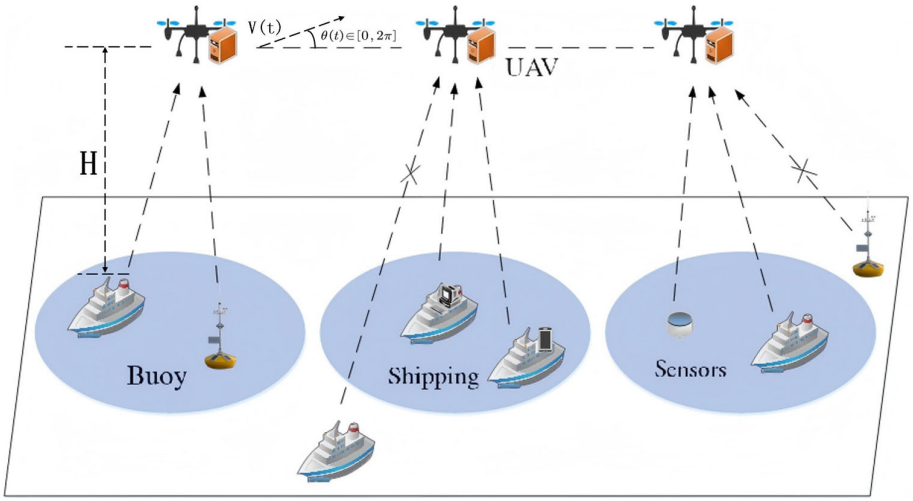


Figure 1. UAV-assisted maritime communication network.

In time slot t, the channel gain between UAV and MIoT device i is:

$$h_i(t) = \alpha_0 d_i^{-2}(t) = \frac{\alpha_0}{\|\mathbf{q}(t+1) - \mathbf{p}_i(t)\|^2 + H^2} \quad (3)$$

In the above equation, α_0 denotes the channel gain when the reference distance is $d = 1$ m.

In the time slot t, the uplink data transmission rate between the MIoT device i and the UAV is calculated as follows:

$$r_i(t) = B\log_2\left(1 + \frac{p_i h_i(t)}{\sigma^2}\right) \quad (4)$$

In the above equation, B is the bandwidth, p_i is the transmission power of the MIoT device, and σ^2 is the noise power.

3.2. Computational Model

Assume each MIoT device has a computational task to be completed within the time period T. In time slot t, the task of the i-th MIoT device can be represented as $\Psi_i = \{d_i, c_i, o_i, t_{i,max}\}$. Here, d_i denotes the size of the computational task, c_i represents the number of CPU cycles required to process 1 bit of data. It is evident that the size of the computation result, denoted by o_i, is typically considerably smaller than that of d_i. The $t_{i,max}$ denotes the maximum tolerable latency, which means the total latency T_i must not exceed $t_{i,max}$. The definition of total latency T_i is given in the following Formula (12).

In the aforementioned system, MIoT devices employ partial offloading technology. Define $\alpha_i(t)$ as the percentage of the i-th MIoT task uploaded to the server, then $1 - \alpha_i(t)$ represents the remaining task ratio executed locally by the MIoT device. In the formula, $\alpha_i(t)$ satisfies $\alpha_i(t) \in [0, 1]$.

(1) Local computing model

In the event of an MIoT device electing to execute a task in a local capacity, the task completion process is rendered autonomous of the UAV edge server. The local computation delay t_i^l of MIoT device i in time slot t can be expressed as:

$$t_i^l(t) = \frac{(1-\alpha_i(t))d_i c_i}{f_i} \tag{5}$$

In the above equation, f_i denotes the computing power of the i-th MIoT device.

Further, the local computing energy consumption corresponding to the i-th MIoT device in time slot t can be expressed as:

$$E_i^l(t) = k(f_i)^2(1-\alpha_i(t))d_i c_i \tag{6}$$

In the above equation, k denotes the calculated energy efficiency factor, which depends on the effective switching capacitance of the chip architecture.

(2) Edge Computing Model

In the event that a component of the MIoT device's task is executed by the UAV, the total latency experienced during this execution encompasses three key components. Firstly, there is the communication time required to transfer the task to the UAV. Secondly, there is the delay associated with executing the computational task on the UAV itself. Finally, the time necessary for the transfer of the computation result back to the MIoT device must be taken into account. It is evident that the magnitude of the computation result is ordinarily considerably less than that of the input data size. Consequently, the transmission delay associated with the transmission of the computation result to the MIoT device is invariably negligible.

In time slot t, the transmission delay for the i-th MIoT device to transfer the task to the UAV server can be expressed as:

$$t_i^{up}(t) = \frac{\alpha_i(t)d_i}{r_i(t)} \tag{7}$$

Accordingly, the energy consumption of the i-th MIoT device in time slot t to transmit the task to the UAV edge server can be expressed as:

$$E_i^{up}(t) = \frac{p_i\alpha_i(t)d_i}{r_i(t)} \tag{8}$$

The processing delay of the i-th MIoT device task on the UAV in time slot t is given by the following equation:

$$t_i^o(t) = \frac{\alpha_i(t)d_i c_i}{f_{uav,i}} \tag{9}$$

In the above equation, $f_{uav,i}$ denotes the computing power allocated to the i-th MIoT device by the edge server carried by the UAV.

The energy consumed by the UAV's flight in time slot t can be expressed as:

$$E_{fly}(t) = \phi\|v(t)\|^2 \tag{10}$$

In the above equation, $\phi = 0.5M_{uav}t_{fly}$ and M_{uav} are related to the load of the UAV and t_{fly} is the fixed flight time.

It can thus be concluded that, within the designated time slot, the energy consumption of the UAV is directly proportional to the computational demands placed upon the edge server when it is operating on the UAV. This relationship can be expressed as:

$$E_{uav}^{o}(t) = P_{uav}\frac{\alpha_i(t)d_i c_i}{f_{uav,i}} = k(f_{uav,i})^2\alpha_i(t)d_i c_i \tag{11}$$

P_{uav} is the transmitted power of UAV in time slot t.

In summary, the total latency of the i-th MIoT device in time slot t can be expressed as:

$$T_i(t) = \max\left\{\left(t_i^{up}(t) + t_i^{o}(t)\right), t_i^{l}(t)\right\} \tag{12}$$

Correspondingly, the total energy consumption of the i-th MIoT device in time slot t can be expressed as:

$$E_i(t) = E_i^{l}(t) + E_i^{up}(t) \tag{13}$$

Furthermore, the overall energy expenditure of the UAV during time slot t can be delineated as follows:

$$E_{uav}(t) = E_{fly}(t) + E_{uav}^{o}(t) \tag{14}$$

All in all, by adjusting the proportion of tasks for local computing Equation (5) and edge computing Equations (7) and (9), it directly affects the composition of total latency. When task complexity is high, increasing $\alpha_{\mathrm{i}}(t)$ reduces local computing pressure; when channel quality is poor, decreasing $\alpha_{\mathrm{i}}(t)$ reduces transmission latency. Channel gain Equation (3) and transmission rate Equation (4) are also affected by changing the distance between the UAV and MIoT devices Equation (2). Optimized trajectories shorten communication distance, reducing transmission latency Equation (7), especially when devices are scattered. Directly determines edge processing latency Equation (9) by allocating edge computing power. When the offloading ratio is high, increasing $f_{uav,i}$ accelerates edge processing but is limited by total server capacity (the following text refers to constraint $C4$), requiring dynamic balancing among multiple devices.

3.3. Problem Description

The model is used to formulate an optimisation problem for a UAV-assisted maritime edge computing network system. This paper jointly optimises the task offloading ratio $\alpha = \{\alpha_i(t)\}$, UAV computational resource allocation $\mathbf{f} = \{f_{uav,i}\}$, and UAV mobility variable $\mathbf{q} = \{\mathbf{q}(t) = [x(t), y(t)]\}$ to minimize the maximum processing latency of all MIoT devices, while satisfying the constraints of maximum available energy consumption $E_{uav,\max}, \forall i$ and tolerable latency $t_{i,\max}, \forall i$ [22]. The corresponding optimization problem can be expressed as:

$$\begin{aligned}\min_{\boldsymbol{\alpha},\mathbf{f},\mathbf{q}} \sum_{t=1}^{T}\sum_{n=1}^{N} T_i(t) \quad & s.t. C1: 0 \leqslant \alpha_i(t) \leqslant 1, \forall i C2: 0 \leqslant x(t+1) \\ & \leqslant x(t) + v(t)t_{fly}\cos\theta(t), \forall t C3: 0 \leqslant y(t+1) \\ & \leqslant x(t) + v(t)t_{fly}\sin\theta(t), \forall t C4: 0 \leqslant f_{uav,i} \\ & \leqslant \mathrm{F}_{\max}, \forall i C5: \sum_{i\in\mathcal{N}} f_{uav,i} \leqslant \mathrm{F}_{\max} C6: T_i(t) \\ & \leqslant t_{i,\max}, \forall i C7: \sum_{t=1}^{T} E_i(t) \leqslant E_{i,max}, \forall i C8: \sum_{t=1}^{T} E_{uav}(t) \\ & \leqslant E_{uav,max}, \forall i\end{aligned} \tag{15}$$

$F_{\max}, \forall i$ represents the maximum computing power of a single MIoT device, and $F_{\max}$ represents the total computing power of the edge server. Constraint $C1$ denotes the range of values for the computational task offload ratio. In this context, each MIoT is capable

of uploading an arbitrary percentage of tasks to the UAV to perform computation, with the remainder of the tasks being handled by the M-IoT device itself. Constraints $C2$ and $C3$ denote the limits of the UAV's movement range, which limit the range of the UAV's trajectory mainly based on the UAV's flight speed capability. Constraint $C4$ denotes that the allocated computational resources cannot exceed the maximum computational resources. $C5$ stipulates that the total amount of computational resources allocated to each MIoT device must not surpass the collective computational capacity of the designated edge server. $C6$ guarantees that the execution time of each task must not exceed the acceptable delay threshold. $C7$ stipulates that the energy consumption of each MIoT device cannot exceed its available energy reserves. $C8$ stipulates that the energy consumption of the UAV across all designated time slots cannot exceed the maximum capacity of the battery.

4. Research on DDPG-Based Strategy for Joint Computation Offloading and Resource Allocation in Maritime UAV Networks

It is evident that this particular optimization problem is a Nondeterministic Polynomial time (NP)-hard problem, which renders it challenging to solve by employing traditional optimization methods. In order to explore the unknown maritime environment, this paper considers the use of the DRL algorithm. The UAV continues to learn through feedback by continuously trying different actions until the action taken leads to the maximization of the long-term system reward. Firstly, the paper presents a model of the optimization problem as a Markov decision problem. It also puts forward a computational offloading scheme for offshore UAVs. This is based on the DDPG algorithm, the aim of which is to obtain the optimal strategy.

4.1. Markov Decision Process

UAV is an autonomous intelligence of the DRL environment that learns the optimal policy to maximize its reward at each time step. In particular, the UAV learns the policy, performs an action, and generates a reward based on that action. The UAV is capable of navigating a confined environment comprising MIoT devices, subject to specific parameter limitations, including altitude and flight duration.

It is imperative to establish a clear distinction between the concepts of state function, action function and reward function in the context of DRL, as these functions exert a significant influence on the efficacy of optimization processes. Consequently, these elements must be defined according to the system model and the objectives of optimization.

State space: The state space in time slot t can be expressed as follows:

$$s_t = \{E_{uav,re}(t), d_i(t), d_{re}(t), \mathbf{p}_i(t), \mathbf{q}(t)\} \tag{16}$$

In the aforementioned equation, $E_{uav,re}(t)$ denotes the residual energy of the UAV in the t-th time slot, $d_i(t)$ denotes the size of the randomly generated task for each MIoT in the t-th time slot, $d_{re}(t)$ denotes the size of the remaining task that the system needs to complete in the whole time period, $\mathbf{p}_i(t)$ and $\mathbf{q}(t)$ denote the location coordinates of all MIoT devices and UAV, respectively.

Action space: The action of the intelligent body mainly includes the computing task offloading ratio, the computational power of the UAV server, and the flight angle and speed of the UAV. Therefore, the action space can be expressed as:

$$a_t = \{\alpha_i(t), f_{uav,i}(t), \theta(t), v(t)\} \tag{17}$$

In the above equation, $\alpha_i(t)$ denotes the task offloading rate of MIoT device i in time slot t. $\theta(t)$, $v(t)$ denote the flight angle and flight speed selected by the UAV, respectively, and $f_{uav,i}(t)$ denotes the computational power allocated by the UAV server to MIoT device i.

(2) Reward function: the intelligent body improves the computing offloading policy through the learning process, so the reward function is maximized MIoT system by minimizing the latency of the UAV-assisted, and the opposite of the processing latency of the time slot is the reward function, which can be expressed as follows:

$$r_t = -\sum_{n=1}^{N} T_i(t) \tag{18}$$

The core objective of the learning process is to guide the agent towards actions that simultaneously minimize the overall system execution cost and maximize the amount of completed tasks. This dual goal is directly encoded into the design of the reward function. The agent's ultimate aim is to maximize the expected cumulative reward, denoted as r_t, over the operational period. In the event of a restriction being violated, a penalty value is assigned to the reward function r_t. A desirable reward function incentivizes the agent to achieve a higher r_t. As outlined in the following section, the DDPG algorithm is described in detail. Since real-time moving end-users and state-continuous dynamic UAV trajectories can lead to a high degree of continuity in the state and action space of the system, it is difficult to obtain an accurate state transfer probability matrix. Therefore, in the next subsection, a DDPG-based computing offloading policy is proposed to explore the optimal policy in the presence of uncertain state transfer probabilities [23].

4.2. DDPG Algorithm Framework

The DDPG algorithm is proposed based on the Actor–Critic (AC) architecture, which trains control strategies by combining deep neural networks. DDPG approximates the Q-value action function using two networks, where the Actor network learns how to select the action and the Critic network learns how to evaluate the value of the policy. The training of the Actor network and the Critic network is independent of each other. But their outputs are used to influence each other's learning. During training, the Actor network learns how to choose the best action by constant trial and error, while the Critic network learns how to estimate the value of a policy by evaluating the choices made by the Actor network.

In order to enhance the stability of the algorithm, the DDPG algorithm employs a dual network architecture. Specifically, the DDPG algorithm uses a deep neural network to approximate the action value function and the policy function. In each iteration, the DDPG algorithm updates the policy network and action value network based on the current policy and action value function. However, since each update leads to a change in the objective function, it makes the weights of the neural network change very drastically, which leads to unstable training. To solve this problem, the DDPG algorithm introduces an objective network. The target network is a copy of the action function values whose weights remain unchanged in each iteration, while the weights of the main network are updated by means of soft updates. The DDPG algorithm is also characterized by the experience replay mechanism, which is an integral feature of the DQN algorithm. This mechanism facilitates the systematic collection and storage of data generated by the intelligent body during its interaction with the environment. This data is then organized into a data structure referred to as the experience memory. At each learning iteration, the intelligent body randomly selects a number of data samples from the experience memory, which are then used for training purposes. The experience replay mechanism has been demonstrated to facilitate the reuse of previous experience data, thereby reducing sample correlation and circumventing the overfitting problem. This, in turn, enhances the robust-

ness and stability of the model, thus leading to an improvement in model performance and generalization ability.

4.3. DDPG-Based Computing Offloading and Resource Allocation Algorithm for Maritime UAV

In order to obtain a better approximation of the policy function, an algorithm based on the continuous action space DDPG is proposed to obtain the optimal offloading policy in the presence of UAV dynamics, to find the optimal offloading decision, UAV mobility, and resource allocation actions, and to minimize the summed latency of all the offshore MIoT devices. The framework of the DDPG algorithm is shown in Figure 2. The DDPG algorithm comprises three primary modules: the main network, the target network, and the replay memory. The main network generates a_t detailed computing offloading and resource allocation policy by mapping the current state a to the action s_t. The main network is a deep neural network. The main network contains two deep neural networks, the Actor network $\pi(s_t|\theta_\pi)$ and the Critic network $Q(s_t, a_t|\theta_Q)$, and the target network exhibits a structure that is analogous to that of the main network, yet it is distinguished by divergent parameter configurations., which can be represented as $\pi\prime\left(s_t|\theta_\pi^T\right)$ and $Q\prime\left(s_t, a_t\middle|\theta_Q^T\right)$. The replay memory is used to store the experience tuple $\langle s_t, a_t, r_t, s_{t+1}\rangle$. Randomized experience from the repository memory breaks the correlation between small batches of samples.

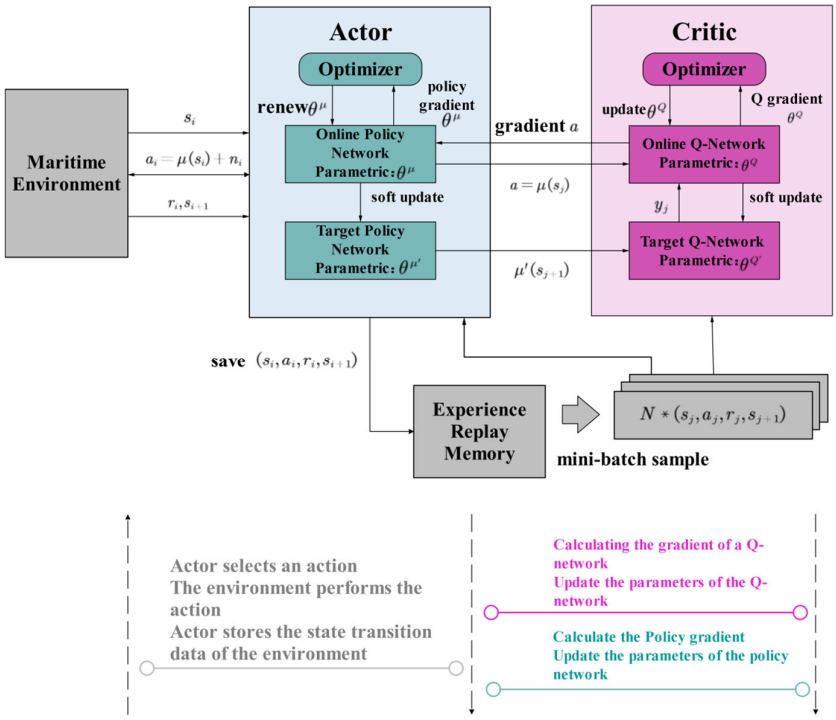


Figure 2. DDPG Framework Diagram.

To solve the exploration–exploitation balance problem, the DDPG algorithm introduces exploration noise. In reinforcement learning, there is a constant balance between exploration and exploitation in order to obtain an optimal policy. While exploiting the existing experience, it is also necessary to explore the unknown states and actions to discover a better policy. The exploration noise in the DDPG algorithm is realized by adding a random noise to the actions of the intelligent body. This noise is usually a Gaussian distributed or evenly distributed random noise, which can increase the exploratory ability of the intelligent body and enable it to better explore the unknown state and action space. The action a_t in state s_t is given by the following equation.

$$a_t = \pi(s_t|\theta_\pi) + \mathcal{N}_t \tag{19}$$

In the aforementioned equation, $\mathcal{N}_t$ is the exploration noise and the addition of appropriate noise can serve to prevent local optimization.

In a similar manner, the DDPG approach employs two target networks with the objective of fixing the target value and enhancing stability. The target action value function, denoted y_t, can be expressed as follows:

$$y_t = r_t + \psi Q\prime\left(s_{t+1}, \pi\prime\left(s_{t+1}\middle|\theta_\pi^T\right)\middle|\theta_Q^T\right) \tag{20}$$

In order to enhance the precision of the critic network, the gradient descent method is employed to minimize the Mean Squared Error (MSE) function between the target action value function y_t and the output of the Critic network $Q(s_t, a_t|\theta_Q)$. The MSE function of the Critic network can be expressed as:

$$Loss(\theta_Q) = \frac{1}{N}\sum_{t=1}^{N}\left(y_t - Q(s_i, a_i \mid \theta_Q)\right)^2 \tag{21}$$

In the above equation, N is the size of the sampling batch.

The core of Actor network is to generate appropriate actions a_t in any state s_t, so as to obtain the maximum state action value function $Q(s_i, a_i)$. Thus, the loss function of Actor network can be expressed as:

$$Loss(\theta_\pi) = -\frac{1}{N}\sum_{t=1}^{N} Q\left(s_t, a_t\middle|\theta_Q\right) \tag{22}$$

The sampled policy gradient can be obtained by minimizing the loss function in (23), and the policy gradient can be expressed as:

$$\begin{aligned}\nabla Loss(\theta_\pi) \approx \frac{1}{N} & \sum_{t=1}^{N}\left[\nabla_{\theta_\pi} Q\left(s, a\middle|\theta_Q\right)|_{s=s_t, a=\pi(s_t|\theta_\pi)}\right] \\ & = \frac{1}{N}\sum_{t=1}^{N}\left[\nabla_a Q\left(s, a\middle|\theta_Q\right)|_{s=s_t, a=\pi(s_t|\theta_\pi)} \nabla_{\theta_\pi}\pi(s|\theta_\pi)|_{s=s_t}\right]\end{aligned} \tag{23}$$

After updating the parameters of the Actor and Critic networks, the network parameters of the two target networks likewise need to be updated, using the following equation:

$$\theta_\pi^T \leftarrow \omega\theta_\pi + (1-\omega)\theta_\pi^T \theta_Q^T \leftarrow \omega\theta_Q + (1-\omega)\theta_Q^T \tag{24}$$

In the above equation, ω is a soft update factor and subject to the following condition $\omega \in [0, 1]$. The proposed DDPG-based computing offloading and resource allocation algorithm is summarized in Algorithm 1. The algorithm first initializes the computing offloading and resource allocation policy $\pi(s|\theta_\pi)$ of the Actor network using parameter θ_π and the action value function $Q\left(s_t, a_t\middle|\theta_Q\right)$ of the Critic network using parameter θ_Q. The parameters θ_π^T and θ_Q^T of their target network are also initialized. Then, based on the current policy $\pi(s|\theta_\pi)$ and state s_t, the Actor network generates an action a_t based on Equation (19). Based on the observed reward r_t and the next state s_{t+1}, a tuple $\langle s_t, a_t, r_t, s_{t+1}\rangle$ is constructed and stored into the experience replay memory. Note that if the experience replay memory is about to be full, the oldest experience samples are deleted to make room for the newest ones. The algorithm is based on the small batch technology and after sampling a portion of the samples, the gradient descent method is exploited to update the Critic network, and the policy gradient is exploited to update the Actor network. After a period of training, the parameters of the target network are updated according to Equation (24).

Algorithm 1 Calculation Offloading and Resource Allocation Optimization Algorithm for Offshore UAV Based on DDPG

1. Initialize the experience replay memory $\mathcal{D}$.
2. To initialize the parameters of Actor Network $\pi(s|\theta_\pi)$, the parameters $\theta_\pi^T \leftarrow \theta_\pi$ of Target Actor Network $\pi\prime\left(s_t\middle|\theta_\pi^T\right)$.
3. To initialize the parameters θ_Q of the Critic network $Q\left(s_t, a_t\middle|\theta_Q\right)$, the parameters $\theta_Q^T \leftarrow \theta_Q$ of the target Critic network $Q\prime\left(s_t, a_t\middle|\theta_Q^T\right)$.
4. for episode = 1, . . ., E do:
5. for t = 1, . . ., T do:
6. To initialize the state of the maritime network environment s_1.
7. Through the current Actor network parameters and behavior noise, select the action $a_t = \pi(s_t|\theta_\pi) + \mathcal{N}_t$.
8. The system performs action a_t, gets rewarded r_t, and moves to the next state s_{t+1}.
9. Deposit of experience samples $\langle s_t, a_t, r_t, s_{t+1}\rangle$ into experience replay memory $\mathcal{D}$.
10. Randomly sample j samples $\langle s_j, a_j, r_j, s_{j+1}\rangle$ from $\mathcal{D}$ to form a small memory of samples.
11. Calculate the Q-value of the target.

$$y_t = r_t + \psi Q\prime\left(s_{t+1}, \pi\prime\left(s_{t+1}\middle|\theta_\pi^T\right)\middle|\theta_Q^T\right)$$

12. Minimize the loss function and update the weight parameters of the Critic network.

$$Loss(\theta_Q) = \frac{1}{N}\sum_{t=1}^{N}\left(y_t - Q\left(s_i, a_i \mid \theta_Q\right)\right)^2$$

13. Update Actor network weights.

$$\nabla Loss(\theta_\pi) = \frac{1}{N}\sum_{t=1}^{N}\left[\nabla_a Q\left(s, a\middle|\theta_Q\right)|_{s=s_t, a=\pi(s_t|\theta_\pi)} \nabla_{\theta_\pi}\pi(s|\theta_\pi)|_{s=s_t}\right]$$

14. Softly update the Critic and Actor target network weights utilizing Equation (24).

$$\theta_\pi^T \leftarrow \omega\theta_\pi + (1-\omega)\theta_\pi^T$$
$$\theta_Q^T \leftarrow \omega\theta_Q + (1-\omega)\theta_Q^T$$

15. end for
16. end for

5. Analysis of Simulation Results

In this section, the effectiveness of the proposed computing offloading and resource allocation algorithm based on the DDPG algorithm is demonstrated in the UAV-assisted maritime MEC system through extensive simulation experiments. Initially, various network parameters appearing in the system are set and adjusted to the optimal parameters. In conclusion, a comparative analysis is conducted between the proposed DDPG algorithm and other baseline methods in order to ascertain the efficacy of the former. It should be emphasized that the DDPG algorithm, by virtue of its capability to handle continuous action spaces along with its experience replay mechanism and target network, is inherently suitable for operation in highly dynamic environments. The simulation experiments in this study fully consider the highly dynamic characteristics of the maritime environment, including factors such as random task generation and dynamic relative positions between

devices and the UAV, laying a foundation for verifying the robustness of the algorithm in dynamic scenarios.

5.1. Simulation Settings

In this part, we use Python 3.12 to build the experimental environment and conduct simulation experiments, in which the task part of the offloading model is built based on the TensorFlow deep learning framework and trained on NVIDIA GeForce GTX 1660 Ti. In the UAV-assisted maritime MEC system, a two-dimensional square area is artificially set up, in which four MIoT devices are randomly distributed in a 100 square meter area, and it is assumed that the UAV is flying at a fixed altitude $H = 100$ m, with the initial takeoff position in the center of the field, the total mass of the UAV is $M_{uav} = 9.65$ kg, and the maximum flight speed is $v_{\max} = 50$ m/s. The whole time period $T = 400$ s is divided into 40 time slots, and the flight time of the UAV in each time slot is $t_{fly} = 1$ s, while the hovering time is $t_h = 9$ s. When the reference distance $d = 1$ m, the channel power gain is set to $\alpha_0 = -50$ dB, the transmission bandwidth is set to $B = 1$ MHz, and the Noise power is $\sigma^2 = -100$ dBm. The tolerable computational latency device of any MIoT device is $t_{i,\max} = 0.5$ s. The task computational complexity, i.e., the CPU cycles required to process each bit, is $c_i = 1000$ cycles/bit. The computational power of the MEC servers of the MIoT device and the UAV are, respectively, set to $f_i = 0.6$ GHz and $f_{uav,i} = 1.2$ GHz. The transmission power of the MIoT device is assumed to be $p_i = 0.1$ W, the available energy to be $E_{i,\max} = 5$ KJ, and the battery capacity of the UAV to be $E_{uav,\max} = 500$ KJ. The detailed simulation parameter settings are shown in Table 1.

Table 1. Simulation experiment parameter settings.

Parameter	Value	Parameter	Value
Number of MIoT devices	4	Energy consumption factor k	10^{-27}
Length and width of sea surface area	100 m	Task computational complexity c_i	1000 cycles/bit
Altitude of UAV flight H	100 m	Transmission power of MIoT devices p_i	0.1 W
Quality of UAV M_{uav}	9.65 kg	Channel gain α_0 when $d = 1$ m	−50 dB
Maximum flight speed of UAV $v_{\max}$	50 m/s	Channel bandwidth B	1 MHz
The flight cycle T	400 s	Noise power σ^2	−100 dBm
Number of time slots	40	local computing power f_i	0.6 GHz
Battery capacity of UAV $E_{uav,\max}$	500 KJ	MEC computing power $f_{uav,i}$	1.2 GHz
Energy available for MIoT devices $E_{i,\max}$	5 KJ	Maximum latency tolerated by MIoT devices $t_{i,\max}$	0.5 s

To determine the optimal hyperparameters for the DDPG algorithm, a systematic experimental paper was conducted, focusing on the learning rates of the Actor and Critic networks. The learning rate critically governs the magnitude of parameter updates during training: excessively small learning rates decelerate parameter updates, whereas excessively large values induce training instability.

As illustrated in Figure 3, the convergence behavior of the proposed algorithm was evaluated under six learning rate configurations, with end-to-end latency serving as the convergence metric. The experimental results demonstrate that: When $\alpha_{Actor} = 0.01$, $\alpha_{critic} = 0.02$, $\alpha_{Actor} = 0.001$, $\alpha_{critic} = 0.002$, and, $\alpha_{Actor} = 0.0001$, $\alpha_{critic} = 0.0002$, the algorithm achieved stable convergence within a bounded latency range, indicating robust convergence.

However, at higher learning rates ($\alpha_{Actor} = 0.1$, $\alpha_{critic} = 0.2$), the algorithm failed to converge globally, settling instead in a suboptimal local solution. This divergence stems from oversized update steps in both networks, which destabilized the training trajectory.

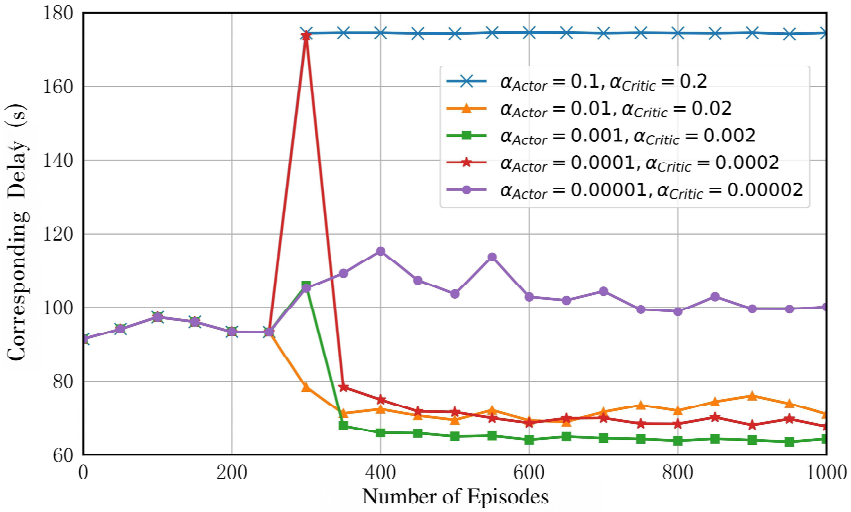


Figure 3. Convergence Performance of the DDPG Algorithm under Different Learning Rates.

Conversely, overly conservative learning rates ($\alpha_{Actor} = 0.00001$, $\alpha_{critic} = 0.00002$) resulted in sluggish convergence, requiring > 1000 iterations to approach stability and exhibiting pronounced oscillatory behavior.

Based on these observations, the optimal learning rates were identified as $\alpha_{Actor} = 0.001$, $\alpha_{critic} = 0.002$, which balanced convergence speed and stability with minimal oscillation.

Next, as illustrated in Figure 4, the impact of varying discount factors γ on the convergence of the proposed DDPG algorithm is investigated. The discount factor determines the significance of future rewards, where a higher discount factor places greater emphasis on future rewards but may also increase training instability. For long-term planning tasks, a higher discount factor (e.g., 0.999 or 0.95) is recommended to better account for the impact of future rewards. Conversely, for short-term decision-making tasks or scenarios where immediate rewards dominate decision-making, a lower discount factor (e.g., 0.1, 0.01, or 0.001) is more suitable. Given the significant environmental variations across different time slots in the system, a lower discount factor yields better performance. As shown in Figure 4, the proposed algorithm achieves optimal performance when $\gamma = 0.001$.

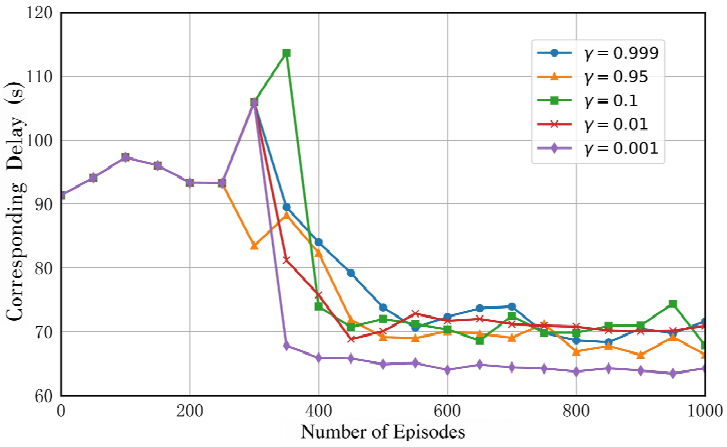


Figure 4. The Proposed DDPG Algorithm under Different Discount Factors.

Finally, the convergence performance of the proposed DDPG algorithm under varying exploration rates is compared, as illustrated in Figure 5. The exploration rate, a hyperparameter, governs the intensity and stochasticity of the action noise. A higher exploration rate expands the spatial scope of the agent's exploration. However, excessively high exploration rates may introduce excessive randomness, hindering the agent from discovering

optimal policies, while overly low exploration rates may lead to insufficient exploration and slow convergence. When the variance (var) is set to 0.5 or 0.1, the exploration rate is excessively high, resulting in nonconvergence. Conversely, $var = 0.001$ corresponds to an insufficient exploration rate, causing sluggish algorithm convergence. As demonstrated in Figure 5, $var = 0.01$ achieves superior performance in the experimental evaluations.

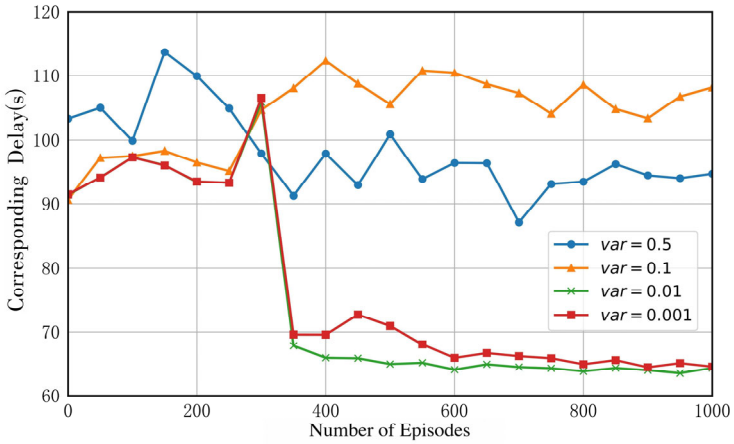


Figure 5. Convergence Performance of the DDPG Algorithm under Different Exploration Rates.

Based on the aforementioned simulation analysis, the neural network parameters employed in the DDPG algorithm are summarized in Table 2.

Table 2. DDPG Parameter Settings.

Parameter	Value
Number of neural network layers	3
Neurons in fully connected layers	[400,300,10]
Actor network learning rate	0.001
Critic network learning rate	0.002
Discount factor	0.001
Exploration rate	0.01
Mini-batch size	64
Total episodes	1000
Time steps per episode	40
Experience buffer size	10,000

5.2. Performance Analysis

The present subsection is concerned with the evaluation of the convergence performance of the DDPG algorithm in comparison with that of baseline algorithms. To evaluate the effectiveness of the proposed DDPG algorithm, we compare its convergence and performance against several relevant baseline methods suitable for this single-agent optimization context: DQN, the foundational AC algorithm, and naive strategies of Local-only and Edge-only computation. These baselines represent common alternative approaches for similar offloading and resource allocation problems. As demonstrated in Figure 6, a performance comparison of different algorithms is illustrated. The two baseline algorithms, Local-only and Edge-only, are contingent on the computational capabilities of MIoT devices and the UAV, which engenders considerable system latency. In contrast, the other three reinforcement learning algorithms initially exhibit unstable latency with large fluctuations.

Following a period of learning, there is a gradual decrease in latency, and the amplitude of fluctuation diminishes.

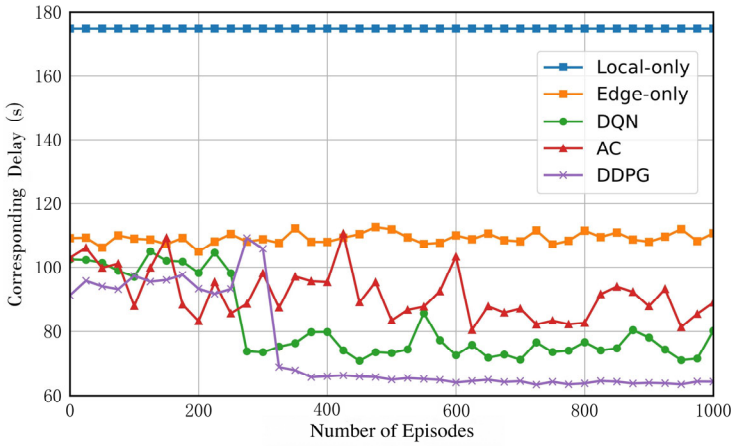


Figure 6. Convergence Comparison of Different Algorithms.

As demonstrated in Figure 6, the AC algorithm does not converge as the number of iterations increases. The behavior of the Actor network is contingent on the value estimation performed by the Critic network. However, the Critic network itself encounters difficulties in achieving convergence. The simultaneous updates of both networks result in unstable training, causing the AC algorithm to diverge. It is evident that both DQN and DDPG adopt dual network architectures, thereby circumventing concurrent updates. However, DQN necessitates discretization of the UAV's continuous action space, a process which readily gives rise to the "curse of dimensionality". It is evident that the DDPG algorithm attains considerably diminished system latency in comparison with the DQN algorithm. This finding serves to underscore the DDPG algorithm's marked superiority in the context of managing continuous state and action spaces.

Figure 7 compares the latency between the proposed DDPG and DQN algorithms under varying local CPU frequencies of MIoT devices. The DDPG algorithm consistently achieves lower latency than DQN across different computational capabilities. DDPG learns a deterministic policy and outputs continuous actions, whereas DQN relies on a limited discrete action set and selects actions via a greedy strategy. Additionally, DDPG stabilizes training by periodically updating target network parameters through soft updates.

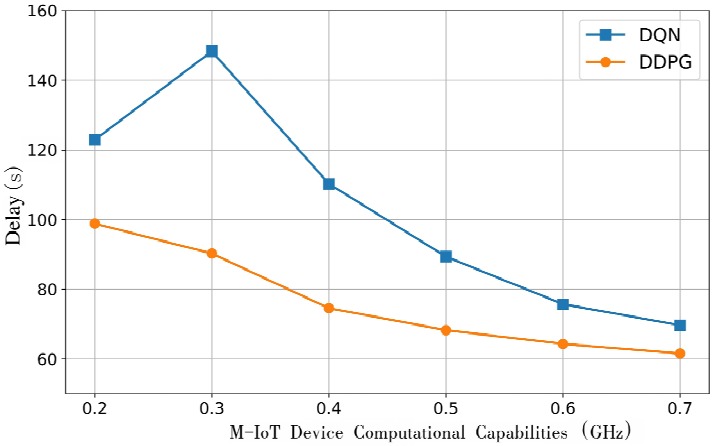


Figure 7. Latency Performance Comparison Under Different MIoT Device Computational Capabilities.

Figure 8 investigates the effect of MEC server computational resources on system latency. The latency of the Local-only algorithm remains unchanged since it does not

offload tasks to the server. As MEC resources increase, the latency of the DDPG-based, DQN, and Edge-only algorithms decreases. The proposed DDPG algorithm consistently outperforms others, highlighting its robustness.

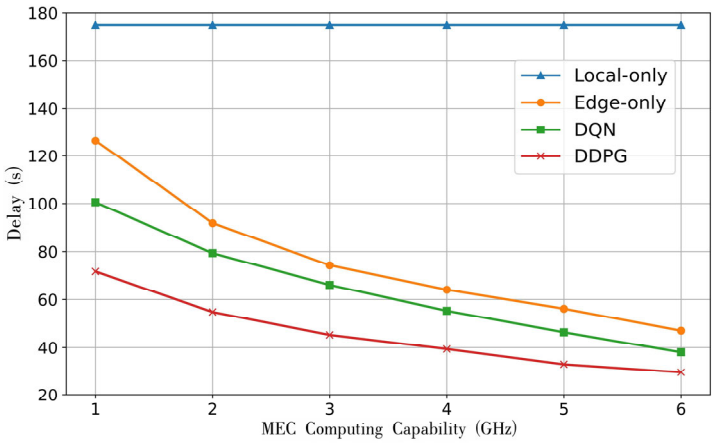


Figure 8. Impact of MEC Server Computational Resources on System Latency.

Figure 9 presents the average processing latency for varying numbers of MIoT devices (1–10). While the latency of DDPG, Local-only, and Edge-only algorithms remains nearly constant, DQN exhibits significant fluctuations due to varying action ranges across different user equipment (UE) quantities. The DDPG scheme achieves the lowest latency by leveraging continuous action policies to optimize convergence.

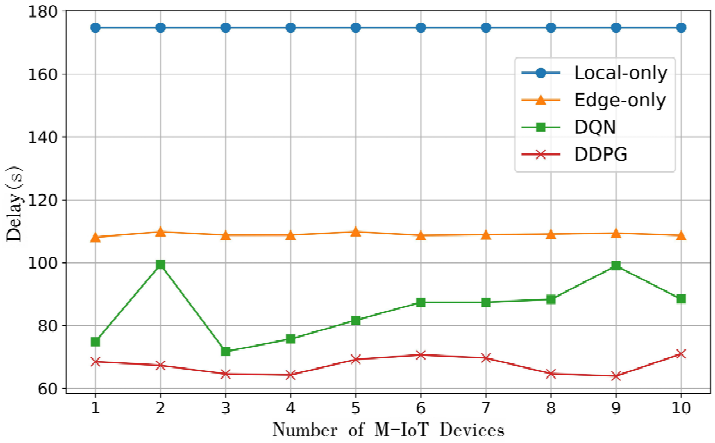


Figure 9. Average Processing Latency Under Different Numbers of MIoT Devices.

Figure 10 compares the performance of algorithms under varying total computational task sizes (60–120 Mbits). For tasks of the same size, DDPG consistently achieves the lowest latency. At 120 Mbits, DDPG reduces system latency to approximately 74 s, outperforming DQN. As task sizes increase, system latency grows accordingly. The Local-only and Edge-only algorithms exhibit higher latency due to inefficient resource utilization. While DQN handles high-dimensional state spaces, its reliance on discrete actions limits its ability to find optimal offloading strategies. In contrast, DDPG efficiently explores continuous action spaces, enabling precise decision-making and optimal policies.

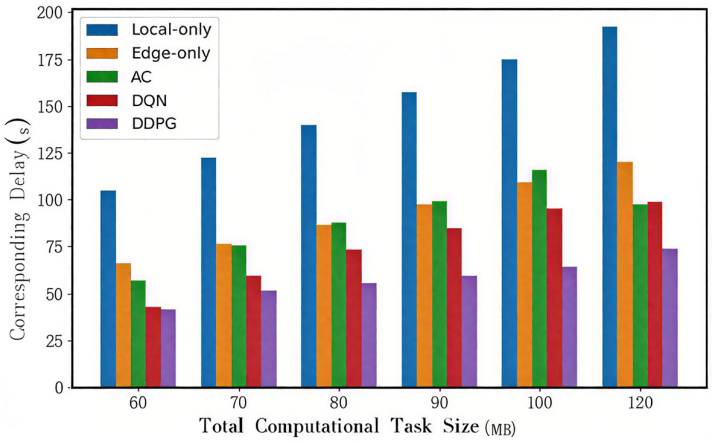


Figure 10. Latency Performance Comparison Under Different Total Task Sizes.

6. Conclusions

This paper investigates a UAV-assisted maritime communication network, formulating an optimization problem to minimize the maximum processing latency of all MIoT devices through joint optimization of computation offloading ratios, UAV mobility, and resource allocation. A DDPG-based maritime UAV offloading algorithm is proposed to obtain the optimal strategy. Comparative experiments with baseline algorithms (e.g., AC and DQN) validate the superiority of DDPG in terms of performance, convergence speed, and latency reduction. The results demonstrate the significant potential of the proposed algorithm in UAV-assisted maritime MEC scenarios. Future work will explore multi-UAV-assisted maritime IoT networks and advanced DRL algorithms for enhanced performance.

While this study presents a DDPG-based approach for optimizing UAV-assisted maritime MEC, it acknowledges limitations including simplified UAV endurance modeling (abstracting dynamic factors like wind), potential impacts of unmodeled maritime channel effects (latency fluctuations, packet loss risk), sensitivity of DDPG convergence to parameters, and a focus on single-UAV scenarios lacking multi-agent coordination. Future work will prioritize extending the framework to multi-UAV networks inspired by fleet potential, enhancing robustness to communication uncertainties, exploring advanced DRL algorithms, and crucially, incorporating real-world maritime data, conducting field validation, and developing more realistic modeling of maritime conditions to address environmental dynamics and communication challenges.

Author Contributions: Conceptualization, Y.X. and Q.Y.; methodology, Y.X.; software, Z.Z.; validation, Q.Y. and Z.Z.; formal analysis, Z.Z.; investigation, Z.Z.; resources, Z.Z.; data curation, Z.Z.; writing—original draft preparation, Z.Z.; writing—review and editing, Z.Z.; visualization, Q.Y.; supervision, Y.X.; project administration, Y.X.; funding acquisition, Y.X. 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 (62271303); The Innovation Program of Shanghai Municipal Education Commission of China (2021-01-07-00-10-E00121).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

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

References

1. Li, K.; Wang, X.; Ni, Q.; Huang, M. Entropy-Based Reinforcement Learning for Computation Offloading Service in Software-Defined Multi-Access Edge Computing. *Future Gener. Comput. Syst.* **2022**, *136*, 241–251. [CrossRef]
2. Lin, Z.; Chen, X.; Chen, P. Energy Harvesting Space-Air-Sea Integrated Networks for MEC-Enabled Maritime Internet of Things. *China Commun.* **2022**, *19*, 47–57. [CrossRef]
3. Borkar, V.S.; Choudhary, S.; Gupta, V.K.; Kasbekar, G.S. Scheduling in Wireless Networks with Spatial Reuse of Spectrum as Restless Bandits. *Perform. Eval.* **2021**, *149–150*, 102208. [CrossRef]
4. Niu, Z. TANGO: Traffic-Aware Network Planning and Green Operation. *IEEE Wirel. Commun.* **2011**, *18*, 25–29. [CrossRef]
5. Wang, J.; Ge, Y. A Radio Frequency Energy Harvesting-Based Multihop Clustering Routing Protocol for Cognitive Radio Sensor Networks. *IEEE Sens. J.* **2022**, *22*, 7142–7156. [CrossRef]
6. Xiao, Z.; Chen, Y.; Jiang, H.; Hu, Z.; Lui, J.C.S.; Min, G.; Dustdar, S. Resource Management in UAV-Assisted MEC: State-of-the-Art and Open Challenges. *Wirel. Netw.* **2022**, *28*, 3305–3322. [CrossRef]
7. Yan, C.; Fu, L.; Zhang, J.; Wang, J. A Comprehensive Survey on UAV Communication Channel Modeling. *IEEE Access* **2019**, *7*, 107769–107792. [CrossRef]
8. Akhtar, M.W.; Saeed, N. UAVs-Enabled Maritime Communications: UAVs-Enabled Maritime Communications: Opportunities and Challenges. *IEEE Syst. Man Cybern. Mag.* **2023**, *9*, 2–8. [CrossRef]
9. Jiao, X.; Chen, Y.; Chen, Y.; Wu, X.; Guo, S.; Zhu, W.; Lou, W. SIC-Enabled Intelligent Online Task Concurrent Offloading for Wireless Powered MEC. *IEEE Internet Things J.* **2024**, *11*, 22684–22696. [CrossRef]
10. Nomikos, N.; Gkonis, P.K.; Bithas, P.S.; Trakadas, P. A Survey on UAV-Aided Maritime Communications: Deployment Considerations, Applications, and Future Challenges. *IEEE Open J. Commun. Soc.* **2023**, *4*, 56–78. [CrossRef]
11. Qiu, T.; Zhao, Z.; Zhang, T.; Chen, C.; Chen, C.L.P. Underwater Internet of Things in Smart Ocean: System Architecture and Open Issues. *IEEE Trans. Ind. Inform.* **2020**, *16*, 4297–4307. [CrossRef]
12. Wei, Z.; He, R.; Li, Y.; Song, C. DRL-Based Computation Offloading and Resource Allocation in Green MEC-Enabled Maritime-IoT Networks. *Electronics* **2023**, *12*, 4967. [CrossRef]
13. Cao, S.; Chen, S.; Chen, H.; Zhang, H.; Zhan, Z.; Zhang, W. HCOME: Research on Hybrid Computation Offloading Strategy for MEC Based on DDPG. *Electronics* **2023**, *12*, 562. [CrossRef]
14. Zhang, J.; Guo, H.; Liu, J.; Zhang, Y. Task Offloading in Vehicular Edge Computing Networks: A Load-Balancing Solution. *IEEE Trans. Veh. Technol.* **2020**, *69*, 2092–2104. [CrossRef]
15. Zhang, K.; Leng, S.; He, Y.; Maharjan, S.; Zhang, Y. Mobile Edge Computing and Networking for Green and Low-Latency Internet of Things. *IEEE Commun. Mag.* **2018**, *56*, 39–45. [CrossRef]
16. Wang, J.; Hu, J.; Min, G.; Zomaya, A.Y.; Georgalas, N. Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 242–253. [CrossRef]
17. Yang, T.; Feng, H.; Gao, S.; Jiang, Z.; Qin, M.; Cheng, N.; Bai, L. Two-Stage Offloading Optimization for Energy–Latency Tradeoff with Mobile Edge Computing in Maritime Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 5954–5963. [CrossRef]
18. Su, X.; Xue, H.; Zhou, Y.; Zhu, J. Research on Computing Offloading Method for Maritime Observation Monitoring Sensor Network. *J. Commun.* **2021**, *42*, 149–163.
19. Jiang, F.; Wang, K.; Dong, L.; Pan, C.; Xu, W.; Yang, K. Deep-Learning-Based Joint Resource Scheduling Algorithms for Hybrid MEC Networks. *IEEE Internet Things J.* **2020**, *7*, 6252–6265. [CrossRef]
20. Wang, R.; Jiang, X.; Zhou, Y.; Li, Z.; Wu, D.; Tang, T.; Fedotov, A.; Badenko, V. Multi-Agent Reinforcement Learning for Edge Information Sharing in Vehicular Networks. *Digit. Commun. Netw.* **2022**, *8*, 267–277. [CrossRef]
21. Lee, Y.H.; Dong, F.; Meng, Y.S. Near Sea-Surface Mobile Radiowave Propagation at 5 GHz: Measurements and Modeling. *Radioengineering* **2014**, *23*, 824–830. [CrossRef]
22. Hasselt, H.V.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-learning. Computer Ence. *arXiv* **2015**, arXiv:1509.06461. [CrossRef]
23. Wang, Z.; Freitas, N.D.; Lanctot, M. Dueling Network Architectures for Deep Reinforcement Learning. JMLR.org. *arXiv* **2015**, arXiv:1511.06581. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.