

Article

HCOME: Research on Hybrid Computation Offloading Strategy for MEC Based on DDPG

Shaohua Cao *, Shu Chen , Hui Chen, Hanqing Zhang, Zijun Zhan and Weishan Zhang 

Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China

* Correspondence: shaohuacao@upc.edu.cn

Abstract: With the growth of the Internet of Things, smart devices are subsequently generating a large number of computation-intensive and latency-sensitive tasks. Mobile edge computing can provide resources in close proximity, greatly reducing service latency and alleviating congestion in mobile core networks. Due to the instability of the mobile edge computing environment, it was difficult to guarantee the quality of service for users. To address this problem, a hybrid computation offloading framework based on Deep Deterministic Policy Gradient (DDPG) in IoT is proposed. The framework is a system consisting of edge servers and user devices. It is used to acquire the environment state through Software Defined Network technologies and generate the offloading strategy by Deep Deterministic Policy Gradient. The optimization objectives in this paper include the total system overhead of the mobile edge computing system, and considering both network load and computational load, an optimal offloading strategy can be obtained to enable users to obtain a better quality of service. Finally, the experimental results show that the algorithm outperforms the comparison algorithm and can reduce the system latency by 20%, while the network load and computational load are also more stable.

Keywords: mobile edge computing; software defined network; deep reinforcement learning; hybrid computation offloading; load balancing



Citation: 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. <https://doi.org/10.3390/electronics12030562>

Academic Editor: Bahman Javadi

Received: 21 November 2022

Revised: 17 January 2023

Accepted: 19 January 2023

Published: 21 January 2023



Copyright: © 2023 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

With the arrival of the 5G smart era, the size of Internet of Things (IoT) devices has increased dramatically, and smart scenarios have become an integral part of our daily lives, such as smart homes, connected cars, and smart healthcare [1,2]. As the number of compute-intensive and latency-sensitive application requests generated by IoT devices continues to increase, resource-constrained IoT devices will face more compute capacity constraints. For example, intelligent Augmented Reality (AR). However, most of the existing AR smart devices can only detect surface phenomena and have insufficient computational power to detect and analyze the nature of complex phenomena in life [3].

To address the problem of lack of computing power, usually cloud computing offloads requests to a central cloud [4]. However, offloading the requests to the cloud for handling also has some problems, i.e., putting all applications on the cloud for execution can easily result in severe network congestion, high energy consumption, load imbalance, high latency, and performance degradation. As an emerging technology, mobile edge computing (MEC) sinks the capabilities of the cloud to the edge of the network, thus providing resources in close proximity to each other, that not only significantly decreases service latency, but also alleviates congestion in the core network [5,6]. However, edge devices are less computationally capable compared to cloud devices, and when faced with massive and complex mobile environments, massive task offloading between edge servers and clouds may result in congestion in the underlying network, and how to coordinate task offloading between cloud, edge servers and mobile smart devices has become a pressing issue.

Software Defined Network (SDN), a flexible, programmable, and scalable network paradigm with clear advantages for coordinating routing and managing devices, has emerged as a viable solution [7,8]. SDN enables flexible network programming by separating the data plane from the control plane [9], using centralized decision making and significantly improving the quality of service (QoS) for users [10,11].

However, SDN-based MEC offload schemes still face some challenges in the IoT. Due to the variability of human activities and the suddenness of requests, the MEC system suffers from a sharp increase in computational load and instability [12]. Therefore, optimizing the decision scheme for computation offloading is a critical issue. In traditional computation offloading strategies, most of the offloading strategies focus on binary offloading strategies [13,14], which means that computational tasks are either executed completely locally on the user devices (UDs) or completely offloaded to the MEC server for execution. However, there are many computational tasks that need to be partitioned into multiple partitions and computed on various devices, for example, when using camera surveillance, deep neural networks are needed to handle video surveillance with real-time, supporting hybrid computational offloading of streaming data are more beneficial for training and inference of deep neural networks [15]. While hybrid computational offloading strategies are suitable for such computational tasks, most of the latest techniques focus only on latency or energy consumption and ignore the computational and network load balancing of the whole system [16,17], which does not guarantee the full utilization of network and computational resources in the system. Moreover, even though a few papers consider load balancing, most of them can only perform binary offloading and not dynamic offloading at arbitrary ratios, which severely bounds the offloading decision [18].

To address the above issues, we propose a Hybrid Computation Offloading for MEC based on the deep deterministic policy gradient framework (HCOME). The framework can execute hybrid computation offloading, allocate computation resources and network resources of MEC system in any ratio, and quickly adapt to dynamic network environment. The total overhead of the MEC system, including total system delay and energy consumption, and computational and network load are taken as the optimization target to improving the stability of the system.

The rest of this paper is presented below. In Section 2, the work related to the application of edge intelligence techniques in IoT is reviewed. Section 3 details the network model and computational offloading decision problem modeling. Section 4 describes the hybrid computational offloading algorithm based on DDPG. Section 5 describes the experimental setup of this paper and shows the experimental results. Finally, Section 6 concludes the paper.

2. Related Work

With the advent of IoT, the demand for storing, accessing, and processing real-time data in the central cloud is growing exponentially, and in the future smart devices generate large amounts of data that will reach ZB order of magnitude [19]. Therefore, redirecting such large amounts of data to the cloud infrastructure will lead to latency issues and may cause network bottlenecks in the future. To address these issues, edge computing, an emerging network paradigm, has emerged as a solution with a bright future to enable mobile access in multiple networks of the IoT.

Morabito et al. [20] reinvented the world of software development with Lightweight Virtualization (LV) techniques by introducing flexibility and new approaches, mainly involving the management and distribution of software. By scaling centralized data centers with massive distributed nodes, edge computing gives virtualization to data sources and end users in close proximity. This emerging paradigm has anytime, anywhere processing power. However, heavy task migration between edge devices and the central cloud can lead to congestion in the underlying network, and wireless communication solutions present new challenges such as mobility management in edge networks [21]. Therefore, to solve this problem, SDN technology is a feasible solution. To address the underlying network

congestion problem, Kaur et al. [22] integrated energy consumption, delay and bandwidth and used SDN techniques for routing and traffic scheduling, which significantly improved the efficiency. Furthermore, Li et al. [23] incorporated SDN technology and edge computing into industrial IoT and proposed an adaptive transmission architecture to find the optimal adaptive power transmission path using different schemes.

In the area of connected cars, in order to solve the problems associated with using high definition (HD) maps to assist driving and improve navigation safety, Peng et al. [24] used SDN technology for unified control and global information through MEC for collaborative processing of computational tasks or storage tasks of autonomous vehicles to improve flexibility. The high latency problem can be effectively reduced by offloading computationally intensive tasks from the vehicle to nearby edge servers [25]. This shows that mobile edge computing is a promising model, which in turn has given rise to vehicle edge computing networks (VECNs). A large number of researchers have combined the advantages of SDNs and MECs to perform related work in terms of task offloading and resource scheduling strategies. For example, Zhang et al. [26] proposed a mobile-aware hierarchical MEC framework in order to address the offloading performance aspects faced by resource-constrained smart devices. The literature [27], on the other hand, uses a dual decomposition approach for goal decoupling for bandwidth and content source optimization.

However, most existing studies on task offloading rarely consider the load balancing of computational resources of edge servers. Many researchers have combined SDN with MEC in issues such as resource allocation, minimizing latency, edge caching, and energy consumption. Singh et al. [28] designed an intent-based network control framework to meet the stringent requirements of data transmission in the underlying network infrastructure. In addition, the above aspects have been studied in the literature [29–31].

Many methods based on deep reinforcement learning (DRL) have been proposed, however, these methods are weak in adapting to new environments because they are inefficient in sampling and require comprehensive retraining to learn update strategies for new environments. To address this challenge, Wang et al. [32] used a meta-reinforcement learning approach for task offloading, by reducing the gradient updates and the number of samples and consequently fast self-adaptation to a new environment. Such a novel offloading approach can reduce the latency by a quarter.

Due to some other limitations of MEC, such as the high cost of infrastructure deployment and maintenance, and the severe pressure on MEC servers due to the complexity and diversity of edge computing environments [33], the appropriate allocation of resources for mobile devices under variable MEC conditions becomes a significant challenge, but there are few works that reflect this. The research goal of this paper is to design a hybrid computational offloading scheme for MEC based on DDPG that can adaptively allocate computational and network resources to achieve overall latency minimization, i.e., effectively reduce the overall computational overhead while ensuring full utilization of network and computational resources in the system.

3. System Model and Problem Formulation

3.1. System Model

In this section, we present an IoT MEC system using SDN technology, as shown in Figure 1, which consists of M MEC servers, N UDs, and an SDN controller. The MEC servers are deployed together with the base stations. The collection of UDs is denoted by $N = \{1, 2, \dots, N\}$. Each MEC server will cover an area of radius L . Each area is an independent system, and each UD in the area will generate a computationally intensive or latency-sensitive task, and when the task needs to be offloaded, the task will be uploaded to this MEC server for processing, thus reducing the processing delay of the task. However, MEC servers are also limited in their computing and storage capabilities, and when UD generates too many tasks, the MEC server does not have enough computational capacity and may not be able to handle all the computational tasks at one time. Therefore, we

suppose that each computational task has the same priority, and a hybrid computational approach is used to process the tasks in the MEC system.

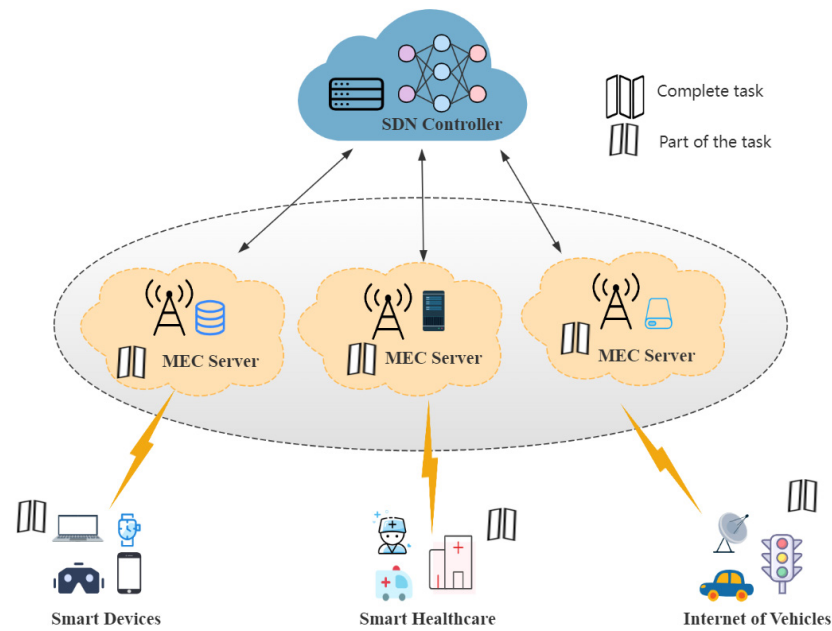


Figure 1. MEC system with SDN technology in IoT.

In addition, the deployment and maintenance costs of MEC servers are high, and it is impractical to deploy MEC servers on a large scale to meet the large number of requests from UD, and the complex and variable edge computing environment can put serious pressure on MEC servers. Therefore, we adopted SDN technology to logically and centrally control infrastructures such as UD and MEC servers to achieve smarter and more efficient network infrastructure management. We have deployed DRL algorithm units on SDN controllers for centralized management and control. First, with the application program interface (API), the DRL module accepts extensive data, including the current network environment status, task status, and infrastructure status. Then, the SDN controller takes the above data as input to the DRL module so that the DRL module can perform training and learning to update the network parameters. Finally, the SDN controller makes an offloading decision according to the offloading policy given by the DRL module.

3.2. Task Model

The task we consider is denoted by $S_i = (C_i, D_i, l_i)$, where C_i denotes the number of CPU cycles needed to complete task S_i , D_i denotes the data size of task S_i , and l_i denotes the maximum tolerated latency of task S_i , i.e., the maximum time needed to complete task S_i . Each task has different parameters. The model in this paper is different from those of other binary computation offloading solutions in that task S_i can perform hybrid computation, i.e., task S_i can be computationally processed locally on UD or offloaded to the MEC server for processing, and also tasks can be offloaded to the MEC server for computation in arbitrary proportions. We denote the computational offloading decision of UD_i as $\omega_i \in [0, 1]$. When $\omega_i = 0$, it means that task S_i of UD_i is computed locally by the device; when $\omega_i = 1$, it means that task S_i of UD_i will be executed on the MEC server; when $0 < \omega_i < 1$, this implies that a mixed computation will be performed, i.e., the ω_i part of task S_i of UD_i is locally processed, and the $(1 - \omega_i)$ part is offloaded to the MEC server for computation.

3.3. Hybrid Computing Model

When task S_i of UD_i needs to perform mixed computation, the ω_i part of data processing delay of task S_i that needs to perform computation locally is

$$T_i^l = \frac{\omega_i C_i}{f_i^l} \quad (1)$$

where $\omega_i C_i$ denotes the number of CPU cycles required for task S_i that is partially locally executed, f_i^l is the computational power of UD_i . In addition, the energy consumption E_i^l of UD_i 's task S_i computed locally is calculated as

$$E_i^l = \omega_i C_i e_i^l \quad (2)$$

where e_i^l is the energy consumed per CPU cycle when task S_i of UD_i is processed locally.

The total latency of processing task S_i when UD 's task S_i for execution is offloaded to the MEC server consists of three parts: the upload latency of uploading data from UD to the MEC server, the processing latency of processing the task by the MEC server, and the download latency of downloading data from the MEC server to UD . Because in comparison with the uploaded data, the size of the downloaded data (execution result) is negligible, the download latency of the data is generally not considered.

Therefore, we define the data upload delay $T_{i,u}^o$ the part of $(1 - \omega_i)$ that needs to be offloaded to the MEC server for calculation is

$$T_{i,u}^o = \frac{(1 - \omega_i) D_i}{r_{i,u}^o} \quad (3)$$

where $(1 - \omega_i) D_i$ denotes the data size of task S_i that is partially offloaded to the MEC server, where $r_{i,u}^o$ is the uplink data rate sent from task S_i to the MEC server from UD_i via the radio access channel, which can be calculated as [34]

$$r_{i,u}^o = W \log_2 \left(1 + \frac{\xi_{i,u} g_i^M}{W \sigma} \right) \quad (4)$$

where W denotes the wireless access channel bandwidth and the wireless access channel bandwidth is distributed equally to each task when there are multiple tasks being offloaded. $\xi_{i,u}$ denotes transmission power for UD_i data upload, g_i^M denotes from the UD to the MEC server for the wireless channel gain, and σ is the channel noise of complex Gaussian.

Meanwhile, the data upload energy consumption $E_{i,u}^o$ in this phase can be obtained from the data transfer power and the data upload delay, denoted as

$$E_{i,u}^o = \xi_{i,u} T_{i,u}^o \quad (5)$$

We define the data processing delay $T_{i,p}^o$ for performing $(1 - \omega_i)$ partial tasks, denoted as

$$T_{i,p}^o = \frac{(1 - \omega_i) C_i}{f_i^o} \quad (6)$$

where $(1 - \omega_i) C_i$ denotes the number of CPU cycles required for task S_i that is partially offloaded to the MEC server, and f_i^o denotes the computational resources allocated to task S_i by the MEC server. We set the total computing resources owned by the MEC server to be F . When there are multiple tasks offloaded to the MEC server, its computing resources will be assigned to each UD task on an as-needed basis, and it should be noted that allocated computing resources of the MEC server to all tasks cannot be more than the total computing resources of the MEC server. The energy consumption $E_{i,p}^o$ for data processing at this stage is

$$E_{i,p}^o = \xi_{i,p} T_{i,p}^o \quad (7)$$

where $\xi_{i,p}$ is the data processing power of the MEC server.

Since the hybrid offload is parallel, the total delay of the hybrid offload is the maximum of the delay computed locally by the ω_i part of task S_i and the delay computed by the $(1 - \omega_i)$ part of the offload to the MEC server. Based on Equations (1), (3) and (6), we can calculate the total latency of task S_i of UD_i when performing the hybrid offload strategy as

$$T_i = \max \left\{ T_i^l, (T_{i,u}^o + T_{i,p}^o) \right\} \quad (8)$$

in terms of energy consumption, the total energy consumption of the hybrid offload should be the sum of the energy consumption computed locally by the ω_i part of task S_i and the energy consumption computed by the $(1 - \omega_i)$ part offloaded to the MEC server, i.e.,

$$E_i = E_i^l + E_{i,u}^o + E_{i,p}^o \quad (9)$$

Thus, the total system expense when task S_i of UD_i performs a hybrid offload strategy can be expressed as

$$\Phi_i = T_i + E_i \quad (10)$$

3.4. Resource Allocation Balancing

When multiple UD tasks are offloaded at the same time, it may cause significant stress to the network environment. When choosing whether to perform offloading, excessive load on the network links and MEC servers may lead to longer processing delays, which in turn affects the offloading decision. Therefore, balancing the network load and computational load in the network while considering the task execution cost when making offloading decisions is important to reduce latency and energy consumption. However, the number of UDs, the processing capacity of MEC servers, and the variation in task size can cause instability in the MEC environment. Our goal is to maintain the stability of the MEC system by balancing the network load and computational load while minimizing the computational overhead under the condition of handling a large number of requests from user devices. Therefore, to reduce network congestion, ensure timely and successful data uploads, and enable users to get better QoS, we define I^{net} as the network load in the system, and the variance of the network load can be expressed as the variance of the network resource allocation and denoted as

$$\text{var}(I^{net}) = \frac{\sum_{i \in K} (I_i^{net} - \frac{\sum_{i \in K} I_i^{net}}{K})^2}{K} \quad (11)$$

where K denotes the number of links between UD and MEC servers. In addition, using I^{cp} to denote the computational load on the MEC server, the variance of the computational load can be expressed as the variance of the computational resource allocation and is denoted as

$$\text{var}(I^{cp}) = \frac{\sum_{i \in N} (I_i^{cp} - \frac{\sum_{i \in N} I_i^{cp}}{N})^2}{N} \quad (12)$$

where N denotes the number of UDs.

3.5. Problem Formulation

We can use Equations (8) and (9) to calculate the total latency and total energy consumption for hybrid offloading and Equation (10) to represent the total system overhead. With the purpose of having better QoS by users, the total overhead of the whole IoT MEC system is minimized as the optimization objective of this paper, i.e., minimizing the task execution time and energy consumption of all UDs in the system, considering both the

network load and the computational load in the system. In this section, b^{net} denotes the variance of the network load and b^{cp} denotes the variance of the computational load. Based on the above network model and hybrid computation model, the optimization problem can be formulated as

$$\min \sum_{i=1}^I \Phi_i + b^{net} + b^{cp} \quad (13)$$

s.t.

$$0 \leq \omega_i \leq 1, \forall i \in I \quad (14)$$

$$f_i^0 \geq 0 \quad (15)$$

$$\sum_{i=1}^I f_i \leq F \quad (16)$$

where condition (14) indicates that each UD task either performs a mixed offload policy, either placing the task entirely at UD for local computation or offloading the task entirely to the MEC server for execution. Condition (15) indicates that the MEC server's computational resources must be allocated to each task, and the allocated resources can be 0. Condition (16) indicates that the sum of computational resources allocated to each task by the MEC server cannot be greater than the sum of the total computational resources of the MEC server. In Equation (13), there are multiple locally optimal solutions in its feasible set that are nonconvex, the time complexity of finding the optimal solution grows exponentially with the number of users when the offload is binary, i.e., 0 or 1. As the number of UDs increases, the size of problem 13 increases rapidly, and thus the nonconvex problem extended from the backpack problem is NP-hard [35]. In the hybrid computational offloading scheme we introduce, the value of ω_i is a continuous domain, solving this NP-hard problem will be more complicated, and using DRL solutions is more suitable for solving such problems [17,35]. Therefore, we propose a DDPG-based hybrid computational offloading scheme that reduces the system overhead of the MEC system. We present the details of our solution in Section 4.

4. Hybrid Computational Offloading Algorithm Based on DDPG

4.1. DDPG Framework

- **State space:** We consider the whole MEC environment as a state space, which mainly includes the state of MEC servers' computing resources, and the network state, the number of UDs and their computing resource states and the state of tasks. However, in practice, considering state of the whole MEC environment leads to a large system overhead, and it gets larger as the number of UDs and MEC servers increases. Therefore, to reduce the additional system overhead, we define the state as

$$S = \sum_{i=1}^I \Phi_i \quad (17)$$

that is, the system overhead for the whole MEC system.

- **Action space:** We use the set of each UD computational offloading proportions as the action space, with the purpose of being able to describe well the computational offloading strategy of all UDs, i.e., $A = \{\omega_1, \omega_2, \dots, \omega_i\}, \omega_i \in [0, 1]$. Since ω_i is a continuous domain, it is easy to choose a reasonable action of arbitrary proportion.
- **Reward function:** After taking the possible actions in each iteration step, the agent of DDPG can obtain a certain reward, which at this point needs to map the goal of our proposed hybrid computational offloading algorithm, i.e., to minimize the total system overhead and also to balance the network load and the computational load in the MEC system. Thus, when setting the reward function, these two objectives should be considered. In addition, to better analyze the merits of the obtained computational

offloading strategy, the reward function needs to become inversely proportional to the total system overhead and load. Thus, the reward function can be expressed as

$$r = -(\alpha_1 T + \alpha_2 E + \alpha_3 b^{net} + \alpha_4 b^{cp}) \quad (18)$$

where T denotes the execution time of the task, E denotes the energy consumption of the executed task, b^{net} denotes the variance of the network load, b^{cp} denotes the variance of the computational load, $\alpha_i, i \in 1, 2, 3, 4$ denotes the weights of each component, i.e., T, E, b^{net} and b^{cp} . In other words, to get a lower delay and lower energy consumption of the MEC system, the reward needs to be bigger and the offloading decision more reasonable.

4.2. DDPG Algorithm

Much of the current research work is implemented by DRL algorithms such as Q-learning, which uses a Q-table to store Q values. Since the action of DRL is a computational offloading decision for all UD, looking up the entire Q-table causes additional overhead when the UD in the MEC system are increasing in size. In contrast, the action space of the DDPG algorithm is a continuous scope [36], while using an empirical buffer to store samples, avoiding the extra overhead of looking up the entire Q-table.

The DDPG framework mainly consists of four deep neural networks, which are the actor network, whose network parameters are denoted as θ^μ , the actor target network, whose network parameters are denoted as $\theta^{\mu'}$, the critic network, whose network parameters are denoted as θ^Q , and the critic target network, whose network parameters are denoted as $\theta^{Q'}$, where the critic network and the actor network are in the same network structure as the corresponding target network.

The actor network selects the current action based on the current state and interacts with the environment to generate the next state and reward. The actor network stores the tuple (s_i, a_i, r_i, s_{i+1}) into the experience buffer as the dataset for training the network. Then, Z sets of data are randomly sampled from the experience buffer, and the target Q-value y_i is evaluated using the actor target network and the critic target network. y_i can be expressed as

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}) | \theta^{\mu'}) | \theta^{Q'} \quad (19)$$

using the mean square error to represent the loss function, which makes the parameter learning process of critic networks more stable and easier to converge [37], the loss function L is denoted as

$$L = \frac{1}{Z} \sum_{i=1}^Z (y_i - Q(s_i, a_i | \theta^Q))^2 \quad (20)$$

the critic network parameters θ^Q are updated using the Adam optimizer [38]. The policy gradient of the actor network is represented as [39]

$$\nabla_{\theta^\mu} J = \frac{1}{Z} \sum_{i=1}^Z (\nabla_a Q(s, a | \theta^Q) |_{s, \mu(s_i)} \nabla_{\theta^\mu} u(s | \theta^\mu) |_{s_i}) \quad (21)$$

the actor network parameters θ^μ are updated using the Adam optimizer. Finally, the target network is updated using a soft update, which is denoted as

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned} \quad (22)$$

where $\tau \in [0, 1]$ is the update factor. The algorithm flow of DDPG is shown in Algorithm 1.

Algorithm 1 DDPG Algorithm

- 1: Randomly initialize the actor network $\mu(s|\theta^\mu)$ and the critic network $Q(s, a|\theta^Q)$;
- 2: Initialize the associated target networks with weights $\theta^{\mu'} \leftarrow \theta^\mu$ and $\theta^{Q'} \leftarrow \theta^Q$;
- 3: Initialize the replay buffer R ;
- 4: **for** each episode **do**
- 5: Randomly generate an initial state s_1 with an action A ;
- 6: **for** each time slot **do**
- 7: Determine the computation offloading strategy according to the actor network and the OU noise $a_t = \mu(s_t|\theta^\mu) + N_t$;
- 8: Execute action a_t at the user agent, and then get reward r_t and observe the next state s_{t+1} ;
- 9: Save the tuple (s_i, a_i, r_i, s_{i+1}) into the replay buffer R ;
- 10: Randomly sample a mini-batch I from the replay buffer R ;
- 11: Update the critic network $Q(s, a|\theta^Q)$ by minimizing the loss L with (20);
- 12: Update the actor network $\mu(s|\theta^\mu)$ by the sampled policy gradient based on (21);
- 13: Soft update the target networks by (22);
- 14: **end for**
- 15: **end for**

The DDPG algorithm is used on the DRL algorithm unit deployed on the SDN controller. Information about the environment across the edge network is collected by the SDN controller and sent to the DRL algorithm module. For the neural network used in the DRL algorithm unit, we use a four-layer fully connected neural network with two hidden layers, and the neural network uses the Relu function as the activation function for the hidden layer. Actor network has a one-dimensional input layer for inputting the state of the MEC system. The output layer of the actor network is N-dimensional, and a Sigmoid function is used to bind the final output of the actor network, which is used to output the computational offloading strategy of each UD in the MEC system. For the critic network, the input layer consists of a computed offloading strategy for each UD and an overall MEC system state. The output layer is one-dimensional and is used to output the Q-value. After that, the SDN controller forwards the final offload strategy obtained by the DRL algorithm unit to each MEC server for offloading through the control plane function with control information. The detailed framework of the DDPG is given in Figure 2.

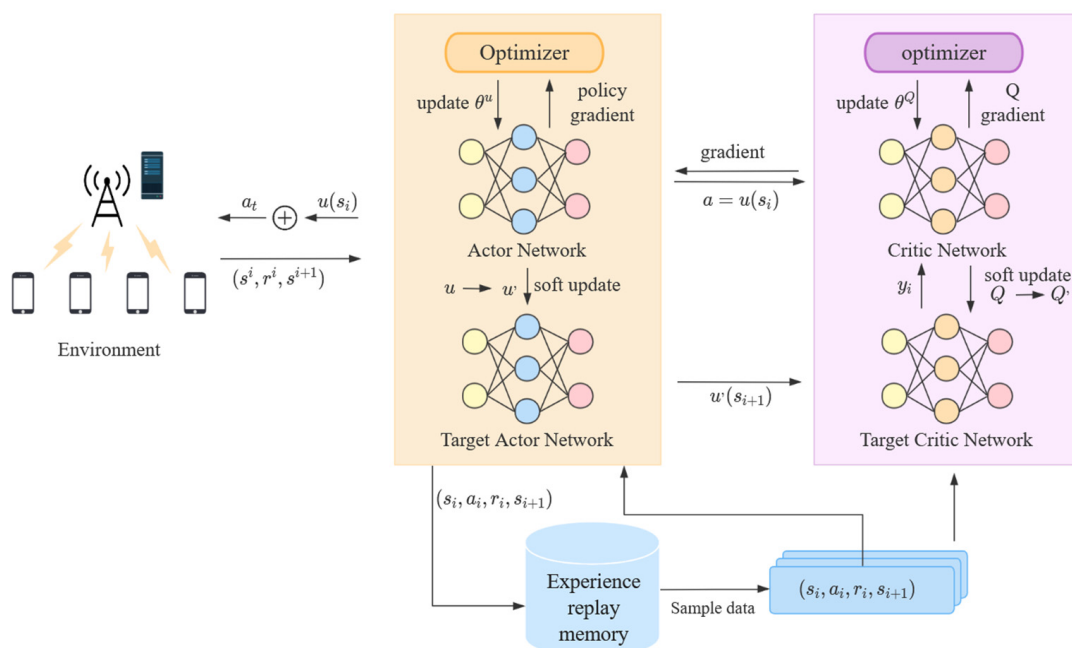


Figure 2. Detailed framework of the DDPG algorithm.

5. Performance Evaluation

5.1. Experimental Setup

In this section, we give the experimental setup and numerical results of a hybrid computational offloading algorithm based on DDPG. To demonstrate the performance benefits of HCOME, it is compared with a fully local computation policy scheme, a fully edge offloading policy scheme, a discrete computational offloading scheme using Q-learning, and the DDPG scheme proposed by [16].

We use the Python programming language to implement a hybrid computational offloading algorithm based on DDPG in the IoT. In our simulated environment, the coverage area d of the MEC server is 200 m, and we set up five users uniformly distributed within the coverage area of the MEC server, each with a task to be processed. The number of CPU cycles C_i required for each UD's task is uniformly distributed between 900 M and 1100 M, and its task data size D_i is uniformly distributed between 300 kbit and 500 kbit. Each UD's computational power is $f_i^l = 1$ GHz/s and the MEC server's computational power is $F = 5$ GHz/s, in the wireless channel, the bandwidth is $W = 10$ MHz, when the UD uploads data to the MEC server, its transmission power is $\xi_{i,u} = 500$ mW [40], the data processing power of the MEC server is $\xi_{i,p} = 100$ mW, the wireless channel gain is $h = d^{-\zeta}$, where $\zeta = 3$, and noise is $\sigma = -174$ dBm/Hz. In addition, we set the maximum tolerated delay l_i of the task to its local processing delay, i.e., if the hybrid computational offloading scheme is not effective in reducing the task completion time, then the task is placed on the UD for local processing to prevent burdening the network. The relevant parameters of the simulation environment and DDPG algorithm we use are shown in Table 1.

Table 1. Simulation environment and DDPG algorithm related parameters.

Parameter	Value	Description
C_i	900 M–1100 M	The number of CPU cycles required for task
D_i	300 kbit–500 kbit	The data size of task
f_i^l	1 GHz/s	The computational power of UD
F	5 GHz/s	The computational power of the MEC server
W	10 MHz	The bandwidth of the wireless channel
$\xi_{i,u}$	500 mW	The transmission power of the UD's data upload
σ	−174 dBm/Hz	The channel noise of complex Gaussian.
actor learning rate	0.0002	Learning rate of actor network
critic learning rate	0.001	Learning rate of critic network
τ	0.005	Update factor for soft update
γ	0.99	Reward discount factor
batch size	64	Sample size in the empirical buffer
buffer size	50,000	Size of the experience buffer

5.2. Experimental Analysis

First, we will evaluate the impact of the number of UDs and the processing capacity of the MEC server for the total system latency. In Figure 3, we investigate the effect of the number of UDs on the total system latency. The horizontal coordinate in the figure is the number of UDs and the vertical coordinate is the total system latency. At this time, the computational capacity of the MEC server is $F = 5$ GHz/s, and the number of cycles C_i required for the tasks of UD is uniformly distributed between 900 M and 1100 M. From the figure, we can see that the total system latency increases significantly as the number of UDs increases. This is because when there are more tasks to be processed using the MEC server, which has limited computational capacity to satisfy the computational requests of all tasks, it generates more computational latency and does not give better QoS to users. More importantly, HCOME has lower latency compared to other schemes, outperforms other schemes. It can reduce the system latency by about 20% compared to the fully local and fully edge computing approaches.

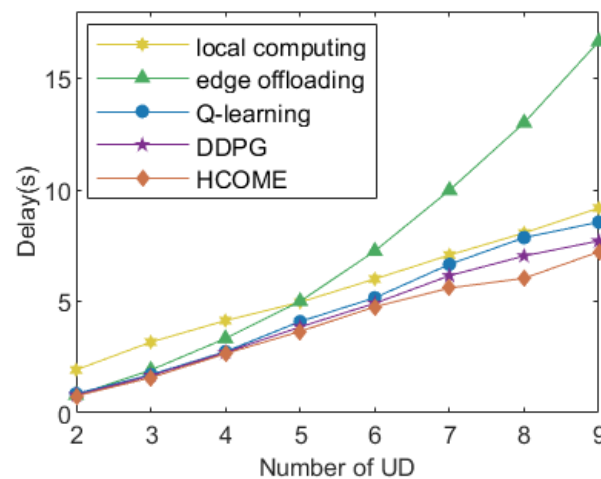


Figure 3. Impact of different UD numbers on latency.

As shown in Figure 4, we investigate how the processing ability of the MEC server affects the total system latency. The horizontal coordinate in the figure is the processing capacity of different MEC servers, and the vertical coordinate is the total system latency. At this time, the number of UDs is $N = 5$, and the number of cycles C_i required for the tasks of UDs is uniformly distributed between 900 M and 1100 M. It can be seen from the figure that the total system latency gradually decreases for all schemes except the local computing scheme as the processing power of the MEC server increases. This is because the fully local computation scheme only needs to use UD's local computing resources instead of putting the tasks on the MEC server for processing, so the latency of the approach using fully local computing is essentially constant and stable. In addition, we can observe that the edge offloading scheme has high latency when the processing power of MEC server does not exceed 5 GHz/s. This is because when the processing power of MEC server is low and there are more tasks offloaded to MEC server, which makes MEC server has insufficient computing power to meet the computation demand of tasks, so the latency of processing tasks is larger. Since the Q-learning algorithm scheme uses discrete offloading, it uses the MEC server for processing tasks while also using local execution of some of the tasks. As a result, as the processing power of the MEC server increases, the edge offloading scheme, the Q-learning algorithm scheme and the DDPG scheme gradually approach HCOME but are always higher than HCOME. Thus, our algorithm outperforms other algorithms.

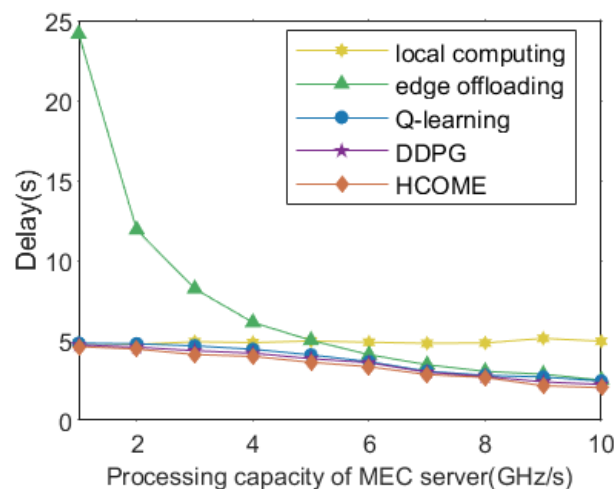


Figure 4. Impact of MEC server's processing power on latency.

Second, we also analyzed the impact of the number of UD and the processing capacity of the MEC server on the total system energy consumption. As shown in Figure 5, we investigated the effect of the number of UD on the total system energy consumption. The horizontal coordinate in the figure is the number of UD and the vertical coordinate is the total system energy consumption. At this time, the computational capacity of the MEC server is $F = 5$ GHz/s, and the number of cycles C_i required for the tasks of UD is uniformly distributed between 900 M and 1100 M. From the figure, we can see that the total system energy consumption of all five schemes gradually increases as the number of UD increases. The DDPG scheme, which considers only the latency performance, results in its excessive energy consumption, which is not conducive to the utilization of resources. On the other hand, the edge offloading scheme consumes the least amount of energy because the edge offloading scheme executes the tasks of UD completely on the MEC server and without using the computational resources on the UD. Compared to executing on UD, the energy consumption of executing tasks on MEC server is lower due to the larger computational power of MEC server. Moreover, we can observe that HCOME is the best execution scheme except the edge offloading scheme, mainly because HCOME allows better and customized task allocation, which makes the energy consumption lower.

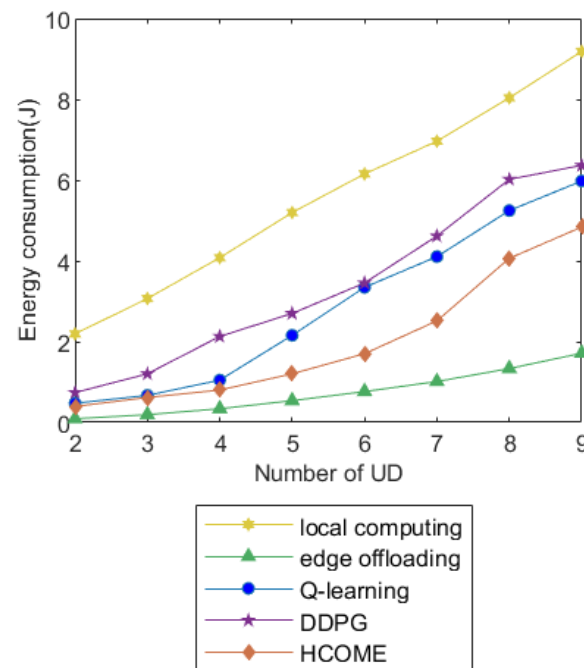


Figure 5. Impact of different UD numbers on energy consumption.

As shown in Figure 6, we investigate the effect of the processing capacity of the MEC server on the total system energy consumption. The horizontal coordinate in the figure is the processing capacity of different MEC servers, and the vertical coordinate is the total system energy consumption. At this time, the number of UD is $N = 5$ and the number of cycles C_i required for the tasks of UD is uniformly distributed between 900 M and 1100 M. From the figure, it can be seen that as the processing power of the MEC server increases, the total system energy consumption gradually decreases for all schemes except the local computing scheme, for reasons similar to the effect of the processing power of the MEC server on the total system latency. It can also be observed that the local computing scheme consumes the most energy, which is due to the fact that the local UD has less computing power and consumes more energy when performing tasks. In addition, we can observe that HCOME is a better solution to dynamically adapt to the environment by adaptively adjusting the distribution ratio of tasks according to the state of the network environment.

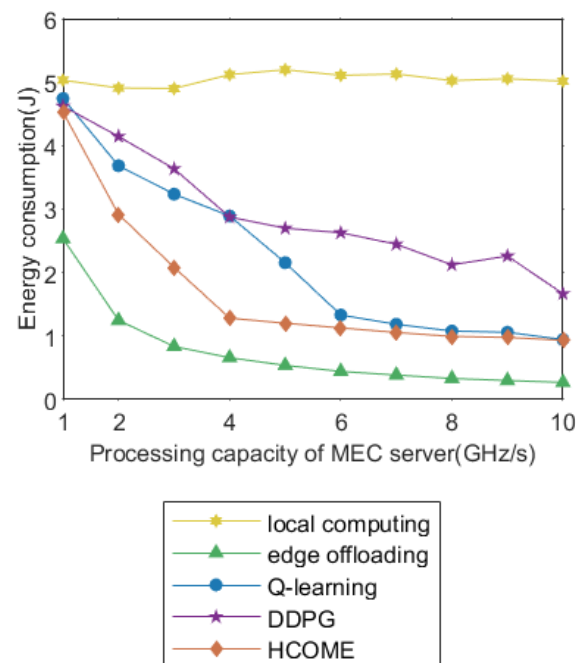


Figure 6. Impact of the processing power of MEC servers on energy consumption.

In the following, we will analyze the balance of the MEC system in terms of the number of UD, the processing power of the MEC server, and the number of CPU cycles required for the tasks of the UD. For the analysis of network load and computational load, we will not compare local computation and edge offloading. First, Table 2 represents the impact of the number of UD with offload tasks on the load variance. As can be seen from the table, the network load and computational load of the HCOME and Q-learning scenarios gradually increase as the number of UD with offloading tasks decreases. This is because when the number of tasks to be offloaded decreases, the MEC server can allocate more computational resources to the tasks to be offloaded, and therefore the computational load increases. In addition, we can see that the trend of change is more pronounced and fluctuates more for the Q-learning solution compared to HCOME. On the other hand, since the DDPG solution only considers the latency performance, it does not balance the computational load and network load in the system, making the computational load and network load too large and extremely unstable. From the above, it is clear that HCOME can better stabilize the MEC system and make the best resource allocation decision when making decisions.

Table 2. Impact of the number of UD with offloading tasks on the load variance.

Number of UD		Number of UD to Be Offloaded	$\text{var}(I^{net})$	$\text{var}(I^{cp})$
HCOME	5	5	0.2944	0.0776
	5	4	0.3252	0.0960
	5	3	0.3384	0.1408
Q-learning	5	5	0.3103	0.0845
	5	4	0.5161	0.1316
	5	3	0.6508	0.1693
DDPG	5	5	0.4169	0.3695
	5	4	0.2894	0.5000
	5	3	0.3835	0.7330

Secondly, as shown in Figure 7, we find that as the number increases, the demand for computational resources from the tasks becomes more complex. Since each UD generates a

task that needs to be processed, the number of tasks that need to be processed will increase as the number of UD increases, resulting in a lower computational load variance that can be obtained by HCOME and tends to be stable, which means that when HCOME makes decisions, it can make balanced use of network resources and prevent network congestion caused by too many UD. On the contrary, it can be seen from the figure that the DDPG scheme has a higher computational load and no clear trend of fluctuation and has a poor performance in balancing the computational load.

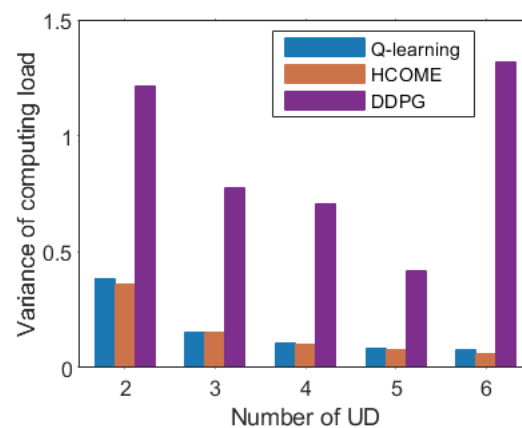


Figure 7. Impact of the number of UD on the computing load variance.

We conducted further experiments to verify the performance of our hybrid computation offloading algorithm in terms of balancing the network load as the processing power of the MEC server increases. As shown in Figure 8, the overall network load becomes more balanced as the processing power of the MEC server increases. This is mainly because the MEC server processing power will lead to a reduction in processing time, which in turn leads to more tasks that can be processed in the same amount of time, resulting in a more balanced network load. On the other hand, the proposed HCOME is always more stable and less volatile than the other two schemes, mainly because our approach not only takes load performance into account when making offload decisions, but also makes resource allocation decisions based on the overall environment and total system overhead of the MEC.

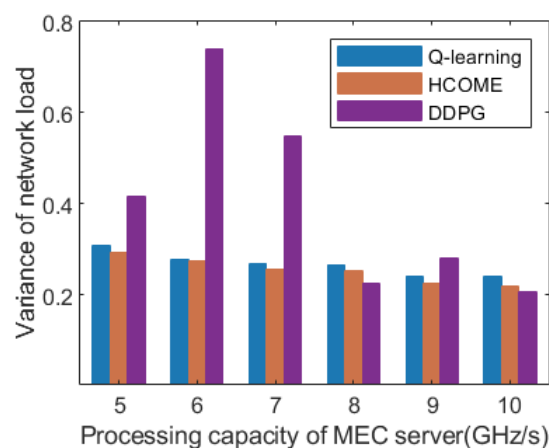


Figure 8. Impact of the processing power of MEC servers on network load variance.

The effect of the number of CPU cycles required for the task on the network load variance is given in Figure 9. As the number of CPU cycles required for a task increases, the network load variance fluctuates slightly, but generally tends to increase. It can be seen from the figure that the network load variance of HCOME is more stable when the number

of CPU cycles is between 900 M and 1050 M, which means that HCOME is very practical and more advantageous when the number of CPU cycles for the task is not too large.

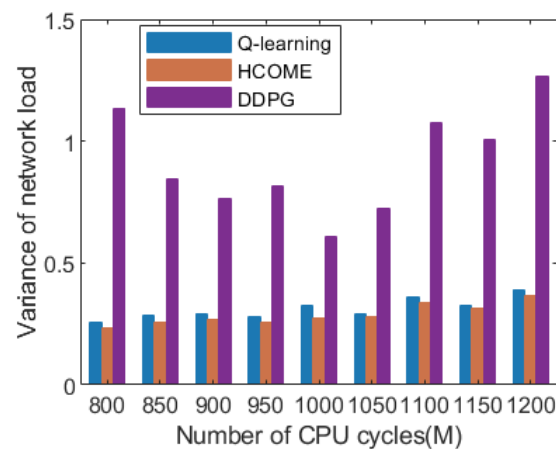


Figure 9. Impact of the number of CPU cycles required for a task on the network load variance.

As shown in Figure 10, we also conducted experiments to verify the convergence of our proposed algorithm. Since only the delay problem is considered in [16] and different factors are considered when setting the reward, only HCOME and Q-learning are compared when verifying the convergence of our proposed algorithm. The horizontal coordinate is the number of iterations and the vertical coordinate is the reward value. At this point, the number of UD is $N = 5$, the computational capacity of the MEC server is $F = 5$ GHz/s, and the number of cycles C_i required for the tasks of UD is uniformly distributed between 300 kbit and 500 kbit. We define the reward as inversely proportional to the total system overhead and load. That is, the larger the reward value, the lower the total system overhead and the more balanced the load of the MEC system, indicating the better performance of the algorithm. It can be seen from the figure that the reward values of both Q-learning algorithm scheme and HCOME increase as the number of iterations increases, and the reward value of HCOME is larger than that of Q-learning algorithm scheme, which has better optimization effect. However, the convergence speed of HCOME is slower than that of the Q-learning algorithm scheme.

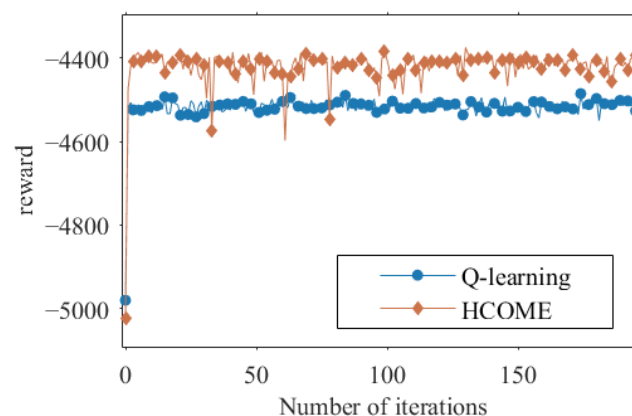


Figure 10. Convergence comparison.

6. Conclusions

In this paper, we study the computational offloading problem in an IoT environment. We consider this problem in terms of both total system overhead and system load and establish a hybrid computational offloading framework based on DDPG, namely HCOME, with an arbitrary offloading ratio per user as the action space. Our algorithm can better

adapt to dynamic MEC environments with burst conditions, reduce system overhead, maintain system stability, and enable users to obtain better QoS. Experimental results show that the HCOME scheme can reduce latency by about 20% compared with traditional fully local computation strategies, fully edge offloading strategies, and Q-learning algorithm strategies, and it is also more stable in terms of network load and computation load, i.e., HCOME has significant advantages in unstable MEC environments. It is thus clear that the DRL solution has great potential compared to other solutions and deserves further investigation. Therefore, in future work, the DRL solution will be further investigated by considering the impact of more complex and dynamic network environments, for example, scaling up experiments, the impact on the performance of solutions when devices randomly join or leave the environment, etc.

Author Contributions: Conceptualization, S.C. (Shaohua Cao), S.C. (Shu Chen), H.C. and W.Z.; methodology, S.C. (Shaohua Cao), S.C. (Shu Chen) and H.C.; software, S.C. (Shu Chen), H.C. and H.Z.; validation, H.Z. and Z.Z.; formal analysis, S.C. (Shaohua Cao) and W.Z.; investigation, S.C. (Shu Chen) and H.Z.; resources, S.C. (Shaohua Cao) and W.Z.; writing—original draft preparation, S.C. (Shu Chen) and H.C.; writing—review and editing, S.C. (Shaohua Cao) and S.C. (Shu Chen); visualization, S.C. (Shu Chen) and Z.Z.; supervision, S.C. (Shaohua Cao). All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Postgraduate Student Innovation Project (YCX2021129).

Data Availability Statement: Not applicable.

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

References

1. Yan, J.; Bi, S.; Zhang, Y.J.A. Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 5404–5419. [\[CrossRef\]](#)
2. Yuan, Y.; Liu, W.; Wang, T.; Deng, Q.; Liu, A.; Song, H. Compressive sensing-based clustering joint annular routing data gathering scheme for wireless sensor networks. *IEEE Access* **2019**, *7*, 114639–114658. [\[CrossRef\]](#)
3. Hu, S.; Li, G. Dynamic request scheduling optimization in mobile edge computing for IoT applications. *IEEE Internet Things J.* **2019**, *7*, 1426–1437. [\[CrossRef\]](#)
4. Premasankar, G.; Di Francesco, M.; Taleb, T. Edge computing for the Internet of Things: A case study. *IEEE Internet Things J.* **2018**, *5*, 1275–1284. [\[CrossRef\]](#)
5. Yu, R.; Li, P. Toward resource-efficient federated learning in mobile edge computing. *IEEE Netw.* **2021**, *35*, 148–155. [\[CrossRef\]](#)
6. Guo, Y.; Zhao, R.; Lai, S.; Fan, L.; Lei, X.; Karagiannis, G.K. Distributed machine learning for multiuser mobile edge computing systems. *IEEE J. Sel. Top. Signal Process.* **2022**, *16*, 460–473. [\[CrossRef\]](#)
7. Rafique, W.; Qi, L.; Yaqoob, I.; Imran, M.; Rasool, R.U.; Dou, W. Complementing IoT services through software defined networking and edge computing: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1761–1804. [\[CrossRef\]](#)
8. Nisar, K.; Jimson, E.R.; Hijazi, M.H.A.; Welch, I.; Hassan, R.; Aman, A.H.M.; Sodhro, A.H.; Pirbhulal, S.; Khan, S. A survey on the architecture, application, and security of software defined networking: Challenges and open issues. *Internet Things* **2020**, *12*, 100289. [\[CrossRef\]](#)
9. Poularakis, K.; Qin, Q.; Nahum, E.M.; Rio, M.; Tassioulas, L. Flexible SDN control in tactical ad hoc networks. *Ad Hoc Netw.* **2019**, *85*, 71–80. [\[CrossRef\]](#)
10. Zarca, A.M.; Bernabe, J.B.; Trapero, R.; Rivera, D.; Villalobos, J.; Skarmeta, A.; Bianchi, S.; Zafeiropoulos, A.; Gouvas, P. Security management architecture for NFV/SDN-aware IoT systems. *IEEE Internet Things J.* **2019**, *6*, 8005–8020. [\[CrossRef\]](#)
11. Akhonzada, A.; Khan, M.K. Toward secure software defined vehicular networks: Taxonomy, requirements, and open issues. *IEEE Commun. Mag.* **2017**, *55*, 110–118. [\[CrossRef\]](#)
12. Tang, F.; Fadlullah, Z.M.; Mao, B.; Kato, N. An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: A deep learning approach. *IEEE Internet Things J.* **2018**, *5*, 5141–5154. [\[CrossRef\]](#)
13. Elgendy, I.A.; Zhang, W.; Tian, Y.C.; Li, K. Resource allocation and computation offloading with data security for mobile edge computing. *Future Gener. Comput. Syst.* **2019**, *100*, 531–541. [\[CrossRef\]](#)
14. Gong, Y.; Wang, J.; Nie, T. Deep Reinforcement Learning Aided Computation Offloading and Resource Allocation for IoT. In Proceedings of the 2020 IEEE Computing, Communications and IoT Applications (ComComAp), Beijing, China, 20–22 December 2020.
15. Chen, C.; Liu, B.; Wan, S.; Qiao, P.; Pei, Q. An edge traffic flow detection scheme based on deep learning in an intelligent transportation system. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 1840–1852. [\[CrossRef\]](#)

16. Deng, X.; Yin, J.; Guan, P.; Xiong, N.N.; Zhang, L.; Mumtaz, S. Intelligent delay-aware partial computing task offloading for multi-user industrial internet of things through edge computing. *IEEE Internet Things J.* **2021**, *80*, 8011–8024. [\[CrossRef\]](#)
17. Tuong, V.D.; Truong, T.P.; Nguyen, T.V.; Noh, W.; Cho, S. Partial Computation Offloading in NOMA-Assisted Mobile Edge Computing Systems Using Deep Reinforcement Learning. *IEEE Internet Things J.* **2021**, *8*, 13196–13208. [\[CrossRef\]](#)
18. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Emerg. Top. Comput.* **2019**, *9*, 1529–1541. [\[CrossRef\]](#)
19. Liu, X.; Yin, J.; Zhang, S.; Ding, B.; Guo, S.; Wang, K. Range-based localization for sparse 3-D sensor networks. *IEEE Internet Things J.* **2018**, *6*, 753–764. [\[CrossRef\]](#)
20. Morabito, R.; Cozzolino, V.; Ding, A.Y.; Beijar, N.; Ott, J. Consolidate IoT edge computing with lightweight virtualization. *IEEE Netw.* **2018**, *32*, 102–111. [\[CrossRef\]](#)
21. Wu, D.; Nie, X.; Deng, H.; Qin, Z. Software Defined Edge Computing for Distributed Management and Scalable Control in IoT Multi-networks. *arXiv* **2021**, arXiv:2104.02426.
22. Kaur, K.; Garg, S.; Aujla, G.S.; Kumar, N.; Rodrigues, J.J.; Guizani, M. Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay. *IEEE Commun. Mag.* **2018**, *56*, 44–51. [\[CrossRef\]](#)
23. Li, X.; Li, D.; Wan, J.; Liu, C.; Imran, M. Adaptive transmission optimization in SDN-based industrial Internet of Things with edge computing. *IEEE Internet Things J.* **2018**, *5*, 1351–1360. [\[CrossRef\]](#)
24. Peng, H.; Ye, Q.; Shen, X.S. SDN-based resource management for autonomous vehicular networks: A multi-access edge computing approach. *IEEE Wirel. Commun.* **2019**, *26*, 156–162. [\[CrossRef\]](#)
25. Zhang, J.; Guo, H.; Liu, J.; Zhang, Y. Task offloading in vehicular edge computing networks: A load-balancing solution. *IEEE Trans. Veh. Technol.* **2019**, *69*, 2092–2104. [\[CrossRef\]](#)
26. 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. [\[CrossRef\]](#)
27. Liang, C.; He, Y.; Yu, F.R.; Zhao, N. Energy-efficient resource allocation in software-defined mobile networks with mobile edge computing and caching. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017; pp. 121–126.
28. Singh, A.; Aujla, G.S.; Bali, R.S. Intent-based network for data dissemination in software-defined vehicular edge computing. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 5310–5318. [\[CrossRef\]](#)
29. Luo, G.; Zhou, H.; Cheng, N.; Yuan, Q.; Li, J.; Yang, F.; Shen, X. Software-defined cooperative data sharing in edge computing assisted 5G-VANET. *IEEE Trans. Mob. Comput.* **2019**, *20*, 1212–1229. [\[CrossRef\]](#)
30. Hou, X.; Ren, Z.; Wang, J.; Cheng, W.; Ren, Y.; Chen, K.C.; Zhang, H. Reliable computation offloading for edge-computing-enabled software-defined IoV. *IEEE Internet Things J.* **2020**, *7*, 7097–7111. [\[CrossRef\]](#)
31. Li, M.; Si, P.; Zhang, Y. Delay-tolerant data traffic to software-defined vehicular networks with mobile edge computing in smart city. *IEEE Trans. Veh. Technol.* **2018**, *67*, 9073–9086. [\[CrossRef\]](#)
32. 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.* **2020**, *32*, 242–253. [\[CrossRef\]](#)
33. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [\[CrossRef\]](#)
34. Luo, Q.; Li, C.; Luan, T.H.; Shi, W. Collaborative data scheduling for vehicular edge computing via deep reinforcement learning. *IEEE Internet Things J.* **2020**, *7*, 9637–9650. [\[CrossRef\]](#)
35. Li, J.; Gao, H.; Lv, T.; Lu, Y. Deep reinforcement learning based computation offloading and resource allocation for MEC. In Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 15–18 April 2018; pp. 1–6.
36. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
37. Peng, H.; Shen, X. Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 2416–2428. [\[CrossRef\]](#)
38. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
39. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the International conference on machine learning, Beijing, China, 21–26 June 2014; pp. 387–395.
40. Cao, Y.; Jiang, T.; Wang, C. Optimal radio resource allocation for mobile task offloading in cellular networks. *IEEE Netw.* **2014**, *28*, 68–73. [\[CrossRef\]](#)

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.