applied sciences

MDPI

Article

Federated Deep Reinforcement Learning for Energy-Efficient Edge Computing Offloading and Resource Allocation in Industrial Internet

Xuehua Li, Jiuchuan Zhang * and Chunyu Pan

Key Laboratory of Modern Measurement and Control Technology, Ministry of Education, Beijing Information Science and Technology University, Beijing 100101, China; lixuehua@bistu.edu.cn (X.L.); chunyupan@bistu.edu.cn (C.P.)

* Correspondence: jiuchuan.zhang@bistu.edu.cn

Abstract: Industrial Internet mobile edge computing (MEC) deploys edge servers near base stations to bring computing resources to the edge of industrial networks to meet the energy-saving requirements of Industrial Internet terminal devices. This paper considers a wireless MEC system in an intelligent factory that has multiple edge servers and mobile smart industrial terminal devices. In this paper, the terminal device has the choice of either offloading the task in whole or in part to the edge server, or performing it locally. Through combined optimization of the task offload ratio, number of subcarriers, transmission power, and computing frequency, the system can achieve minimum total energy consumption. A computing offloading and resource allocation approach that combines federated learning (FL) and deep reinforcement learning (DRL) is suggested to address the optimization problem. According to the simulation results, the proposed algorithm displays fast convergence. Compared with baseline algorithms, this algorithm has significant advantages in optimizing the performance of energy consumption.

Keywords: industrial internet; mobile edge computing; federated learning; deep reinforcement learning; computing offloading; resource allocation

Citation: Li, X.; Zhang, J.; Pan, C. Federated Deep Reinforcement Learning for Energy-Efficient Edge Computing Offloading and Resource Allocation in Industrial Internet. *Appl. Sci.* **2023**, *13*, 6708. http://doi.org/10.3390/app13116708

Academic Editor: Antonio Fernández-Caballero

Received: 26 April 2023
Revised: 28 May 2023
Accepted: 29 May 2023
Published: 31 May 2023

1. Introduction

The amount of data from mobile smart terminal devices in intelligent factories is growing rapidly as the Industrial Internet age progresses [1]. Due to resource limitations of terminal devices such as logistics handling robots, mobile assembly robots, and unmanned delivery vehicles, they cannot process large amounts of data by themselves [2]. In situations with insufficient bandwidth, traditional cloud computing models have difficulty responding to terminal device requests in real time. Traditional centralized processing is overloaded by the severe bandwidth and energy demands of data transfer from the endpoint to the cloud [3]. It has become a development trend to promote the sinking of cloud computing capabilities and improve the processing capabilities of edge devices, which has led to the emergence of MEC in the Industrial Internet [4–6]. Intelligent factory scenarios are more sensitive to energy consumption, which require more data computing and higher data security [7]. MEC has a great effect on saving energy consumption [8]. MEC migrates the original cloud computing resources and services of the intelligent factory to a location closer to the terminal, effectively reducing the energy consumption caused by communication and computing [5]. MEC promotes cloud services on the edge to be closer to mobile smart devices by utilizing the capabilities of edge computing servers. Mobile industrial devices constrained by limited battery capacity and computing power are able to offload compute-intensive tasks to MEC servers [9]. In addition, MEC is more adept at handling tasks that require real-time execution than traditional cloud services [10]. Edge

computing may also be employed in Industrial Internet systems to offload computer tasks and reduce network traffic [11].

MEC is primarily used for real-time processing of large-scale industrial data, but it still faces challenges in communication between mobile devices and MEC servers. This challenge imposes additional costs on the system, including energy consumption and delay. Therefore, in the context of the Industrial Internet, it is crucial to improve resource allocation and unloading strategies, which currently represent a key issue [12]. Moreover, in the field of Industrial Internet, terminal and edge devices have limited computing power. When dealing with complex nonconvex problems, traditional machine learning methods often encounter issues such as nonconvergence or local convergence [13], making the system optimization challenges even more complex. Additionally, factories place high importance on data privacy [14].

In summary, current academic research on edge-terminal collaborative computing offloading and resource allocation for MEC networks has been conducted extensively. However, there is insufficient research on multiedge-terminal collaboration for computing offloading and resource allocation in the intelligent factory of the Industrial Internet scenarios. Therefore, to better facilitate the development of the Industrial Internet and meet business requirements, it is the focus of research to solve the MEC system's multiedge-terminal collaborative computing offloading and resource allocation through cutting-edge machine learning algorithms in the Industrial Internet scenario.

Our paper investigates the intelligent factory within the Industrial Internet MEC setting. We propose an algorithm that merges FL and DRL to optimize the system's overall energy consumption. This optimization is achieved through multiedge-terminal collaborative computing offloading and resource allocation. We consider an intelligent factory with an Industrial Internet multiedge-terminal MEC system, which consists of several small base stations connected to edge servers and several mobile smart industrial terminal devices. Tasks can either be run locally on terminal devices or offloaded to edge servers. The overall energy consumption of Industrial Internet multiedge-terminal MEC systems is the focus of our article. In response to the energy consumption optimization efficiency problem of traditional optimization algorithms in the Industrial Internet scenario with multiedge-terminal collaboration, we design a computing offloading and resource allocation algorithm utilizing deep deterministic policy gradient (DDPG) [15] combined with FL [16]. By jointly optimizing task offloading ratio, subcarrier quantity, transmission power, and computing frequency, a better strategy can be obtained by this algorithm, thus reducing the system's overall energy consumption. This paper makes the following primary contributions:

- This article examines the resource allocation and computing offloading of multiedge-terminal collaboration within the MEC for an Industrial Internet intelligent factory. In this situation, work from terminal devices can be delegated to a single edge server or a group of edge servers simultaneously. Tasks are also capable of being computed locally on terminal devices. Task offloading has the potential to be either full offloading or partial offloading to the edge servers.
- Our focus is on enhancing the energy efficiency of conventional algorithms, and we are exploring ways to lower the system's entire usage of energy. Specifically, we are investigating how multiedge device collaboration can be leveraged to optimize computing offload and resource allocation. We transform the resource allocation and computing offloading combined optimization issue into a Markov decision process (MDP) problem. To minimize energy consumption, we want to optimize the MEC system. We plan to accomplish this by optimizing various system variables, including the task offloading ratio, subcarrier quantity, transmission power, and computing frequency, while satisfying the constraint of task completion delay.
- A novel computing offloading and resource allocation algorithm due to DDPG and FL is introduced in this work. DDPG is an effective DRL algorithm with two deep neural networks. FL is a decentralized machine learning technique that makes devices to learn from each other without sharing their local data. We integrate a federated

learning approach into the policy network and value network of the conventional DDPG algorithm. Following gradient updates, we upload the neural network parameters from multiple DDPG algorithms to a cloud node for federated averaging. Subsequently, these parameters are distributed back to the neural networks of each DDPG algorithm to facilitate the updating of their respective neural network parameters. Combining the DDPG algorithm with FL enables the multiuser system in the Industrial Internet networks to achieve a better strategy for computing offloading and resource allocation with lower overall usage of energy while maintaining computing efficiency.

To provide structure and clarity to the remainder of this paper, we organized it in the following manner. In Section 2, we present related work in the background of the article. In Section 3, we provide the system model and the optimal solution challenge. In Section 4, to tackle the optimization problem we are addressing in this paper, we introduce a computing offloading and resource allocation algorithm. In Section 5, we test and verify the proposed algorithm's performance and compared it against the baseline to assess its effectiveness. Section 6 gives the conclusions of this paper.

2. Related Work

In this section, we introduce the related work on computation offloading and resource allocation in Industrial Internet MEC systems, as well as the relevant work on DRL and FL algorithms.

2.1. Transmission Efficiency

Some studies focus on improving the data transmission efficiency of MEC systems. For instance, Shu et al. [17] proposed an effective task offloading scheme to address access control and task offloading problems in multiple-user MECs, while Wang et al. [18] suggested a stable offloading strategy based on the preferences of specific tasks and edges. At the same time, the performance of such offloading strategies depends on both communication and computing resources, which impact data transfer rates, the energy use of terminal devices, and task processing times. Therefore, it is worthwhile to discuss how to manage communication and computing resources effectively in the Industrial Internet.

In addition, early research on MEC prioritized enhancing the performance of MEC systems. For instance, for the best allocation of resources in MEC, Tran et al. [19] suggested a convex optimization approach. The integrated optimizing for communication, processing resources, and offloading is the subject of some MEC research at the moment. Zhao et al. [20] proposed a game theory methods optimization scheme for edge-terminal collaborative computing offloading and resource allocation, which maximized system utility in a cloud-assisted MEC system. Kai et al. [21] proposed a pipeline-based offloading strategy that uses both mobile smart devices' and edge servers' offloaded computing-intensive tasks to specific edge servers and central cloud servers according to their computing and communication capabilities, respectively. Wang and colleagues [22] developed a cloud–edge–client collaborative edge intelligent layered dynamic pricing mechanism and presented an enhanced two-layer game approach to describe the collaboration. Their study introduced a unique forecasting method based on K-nearest neighbors that improves game convergence and lowers the amount of invalid games. Yaqoob et al. [23] proposed a routing protocol with a multiple-objective cost function to optimize the performance and minimize power consumption in distributed medical detection systems. You et al. [24] investigated the resource allocation problem in a multiple-user MEC on orthogonal frequency division multiple access (OFDMA) and suggested a low-complexity suboptimal algorithm. To determine the appropriate resource allocation, they created an average offloading prioritization function.

Many studies have developed multilayer optimization methods and iterative algorithms to solve the above problems [22,25]. Extracting probable correlations between alterations in the environment and the operations of entities in Industrial Internet scenarios are difficult tasks for algorithms due to their complex and dynamic nature [26]. As a

solution, Xiao et al. [27] introduced mobile offloading schemes on DRL that can enhance computational performance by choosing the best offloading strategy. To reduce the computing cost of the MEC in multiple inputs and multiple outputs, the capabilities of local processing and task offloading may be automatically allotted by learning rules from the local observations of each user on the MEC system. Chen et al. [28] studied a decentralized dynamic computing offloading technique based on DRL. To be able to determine the best offloading and resource allocation for each task in a MEC system that operates under the influence of wireless fading channels and random edge computing, Yan et al. [29] proposed a DRL algorithm that utilizes the actor–critic (AC) learning architecture. Li et al. [30] suggested implementing the DRL algorithm to devise a stochastic scheduling scheme for MEC, aiming to minimize the driving distance of autonomous vehicles. To choose the best cloud or edge server for offloading, Yan et al. [31] created a combined task-offloading and load-balancing optimization technique based on DRL. Nath et al. [32] explored the difficulties of dynamical caches, computing offloading, and allocation of resources in multiple users MEC. To address these challenges, they proposed a dynamic scheduling strategy that uses DRL to solve the problem, especially in situations with random task arrivals. In order to optimize the benefits of IoT applications while guaranteeing equitable resource distribution, AlQerm et al. [33] presented a novel Q-learning system.

2.2. Algorithms

In terms of algorithms, DRL emerged from the work of Mnih et al. [34], who combined convolutional neural networks with the Q-learning algorithm from traditional reinforcement learning to propose the Deep Q-Network (DQN) model. The DQN model was initially applied to visual-perception-based control tasks, representing a groundbreaking achievement in DRL. Building upon this idea, Lillicrap et al. [15,35] expanded the Q-learning algorithm using the concept of DQN and introduced the DDPG algorithm, which combines value and policy gradients to handle the DRL problem in continuous action spaces. On the other hand, the FL algorithm enables collaborative model training without the need for data sharing [16]. In this approach, each device possessing local data keeps its data locally and exchanges undisclosed parameters with other devices to create a shared global model within the FL system, employing techniques such as federated averaging [36]. The processed global model is then distributed to each device and exclusively used for local model training [37]. Yaqoob et al. [38,39] introduced a hybrid framework that combines FL, artificial bee colony optimization, and support vector machines. This framework aims to achieve optimal feature selection and classification of diseases on the client side of health service providers, effectively addressing privacy concerns.

FL is beneficial in reducing convergence problems, improving algorithm performance, and ensuring data privacy. Regarding the integration of DRL and FL, Yu et al. [40] proposed a novel two-timescale DRL approach that utilizes FL for distributed training of the aforementioned model. The primary objective is to minimize overall offload latency and network resource utilization by jointly optimizing computation offloading, resource allocation, and service cache placement, all while safeguarding data privacy for edge devices. In a similar vein, Wang et al. [41] developed a method that leverages deep Q-learning to enhance the efficiency of FL algorithms and reduce the number of communication rounds required for FL. In another work, Khodadadian et al. [42] devised a framework based on FL and reinforcement learning. Multiple local models collaborate to learn a global model, and its convergence is analyzed. Lastly, Zhu et al. [43] proposed a federated reinforcement learning approach that combines the privacy protection offered by FL with the problem-solving capabilities of reinforcement learning. Their method introduces increased concurrency to enhance the learning process. Luo et al. [44] introduce a layered framework for federated edge learning that migrates the model aggregation part from the cloud to the edge server and offers a productive resource scheduling technique that will reduce the overall cost. In order to reduce compute offload latency and resource usage while maintaining data privacy for edge devices, Zhu et al. [45] developed a dual-timescale technique employing

distributed FL training. For AC-based FL, Liu et al. [46] introduced an online approach. The model is trained by each edge node in its MEC system, and a weighted loss function is used to facilitate model optimization.

The above is the related work of our article. At present, academic research on edge collaborative computing offloading and resource allocation of MEC networks has been extensively carried out. However, for a smart factory in an Industrial Internet scenario, the research on multiedge-terminal collaborative computing offloading and resource allocation is not enough. Recognizing this research gap, we initiated an in-depth study to address this crucial area of investigation.

3. System Model

In this section, we introduce and derive a system model and an optimization problem for computation offloading and resource allocation in the context of an Industrial Internet multiedge-terminal MEC. Within this MEC system, the energy consumption comprises the combined energy consumed by all terminal nodes for computation, as well as the energy consumed during transmission to the edge servers.

As illustrated in Figure 1, we propose a multiedge-terminal collaborative MEC system in an intelligent factory. It is assumed that the system consists of M mobile smart industrial terminal devices and N small base stations with edge servers. The index of terminal devices is $i \in \{1, 2, \ldots, M\}$, and the index of edge server is $j \in \{1, 2, \ldots, N\}$. The terminal devices are located on the same production line in the intelligent factory. Each device has a task chain, and the total data volume of the task chain is I_i. Several independent tasks make up the task chain. Each epoch t handles one task, $t \in \{1, 2, \ldots, \mathcal{T}\}$; thus, there are $\mathcal{T}$ epochs in total. $I_{i,t}^{a}$ represents the size of the data for the task to be computed by the terminal devices i in the present epoch t,

$$I_i = \sum_{t=1}^{\mathcal{T}} I_{i,t}^{a}. \tag{1}$$

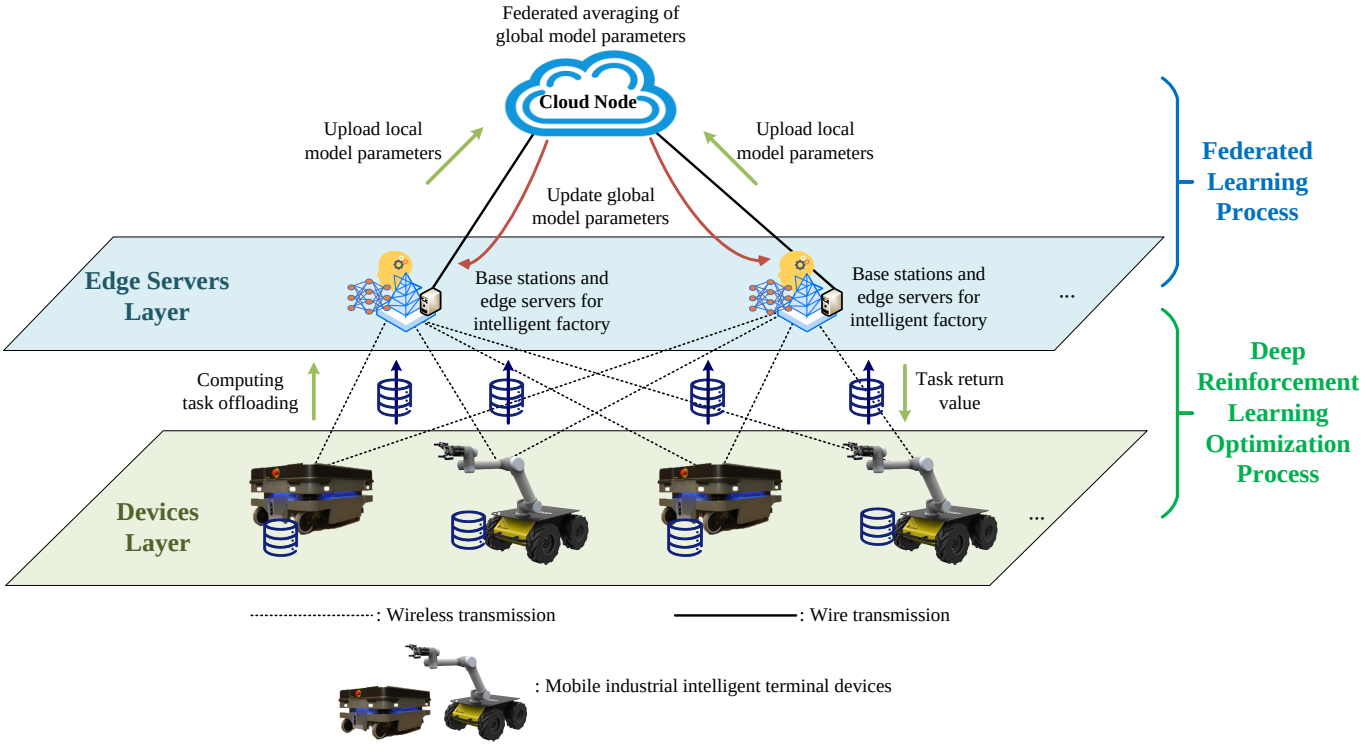

Figure 1. MEC systems for an intelligent factory.

Each task in the task chain has three choices to be computed. Firstly, tasks can be partially or fully computed locally on the terminal device. Secondly, the tasks can be partly or fully offloaded to any single edge server for computing. Thirdly, the tasks can be partly or fully offloaded to multiple different edge servers to be computed simultaneously. Because of the powerful processing capabilities of edge servers, they can simultaneously process tasks offloaded from multiple terminal devices in parallel.

3.1. Communication Model

The transmission process only takes place when the task is offloaded. The transmission process uses OFDMA, which is able to occupy multiple subcarriers simultaneously when the terminal device offloads data. Therefore, in epoch t, the data uplink transmission rate $R_{i,j,t}$ for the terminal device i sending tasks to the edge server j is given by

$$R_{i,j,t} = w_{i,j,t} B \log_2 (1 + \frac{h_{i,j,t} p_{i,j,t}}{\delta^2}), \tag{2}$$

where $w_{i,j,t}$ is the number of subcarriers that have been selected by the terminal device for the data offloaded, B denotes the bandwidth of the subcarrier, $p_{i,j,t}$ denotes the transmission power of each subcarrier, $h_{i,j,t}$ denotes the channel gain of each subcarrier, and δ^2 is the channel noise power. The subcarrier channel gain $h_{i,j,t}$ can be expressed as

$$h_{i,j,t} = \alpha_0 d_{i,j,t}^{-2}, \tag{3}$$

where the channel gain at a reference distance of one meter is denoted by α_0, and $d_{i,j,t}$ is the Euclidean distance between the terminal device i and the edge server j, which is set at random according to a uniform distribution.

In MEC systems, the size of the data provided by the edge server as a result of the computations is usually so small as to be negligible [12]. Therefore, the transmission delay of the downlink is not considered. The transmission delay of the uplink is given by

$$T_{i,j,t}^{trans} = O_{i,j,t} \frac{I_{i,t}^a}{R_{i,j,t}}, \tag{4}$$

where $O_{i,j,t}$ denotes the offloading ratio of the terminal device i to the edge server j. The energy required to offload a task from the terminal device to the edge server is equal to the energy consumed by the terminal device i to transmit data to the edge server j. Then, the energy consumed by the terminal device during transmission can be expressed as

$$E_{i,j,t}^{trans} = w_{i,j,t} p_{i,j,t} T_{i,j,t}^{trans}. \tag{5}$$

The edge server is fixed, as it is wired to the base station. Data can be transferred between multiple edge servers over optical fiber links with negligible delay and energy loss during transmission.

3.2. Computation Model

Tasks that have been offloaded are processed in parallel by the edge server upon reception. Due to the edge server's relatively high computational performance [5], it is assumed that the delay consumed in processing tasks and waiting in the serial queue of tasks is not considered. Setting the terminal device's central processing unit (CPU) has a multicore function. The device utilizes multicore computing to process multiple tasks concurrently, allowing for adjustable computing frequencies for core $k \in \{1, 2, \ldots, K\}$. Additionally, the regulation of its computing frequency is continuous. The computing frequency $f_{i,k,t}$

of the core k is in the frequency range $F \in (f^{\min}, f^{\max})$. According to the offloading ratio, the size of the task data computed locally on the terminal device i is given by

$$I_{i,t}^{u} = \left(1 - \sum_{j=1}^{N} O_{i,j,t}\right) I_{i,t}^{a}. \tag{6}$$

Define the size of the task data allocated to the core k in the terminal device i as $I_{i,k,t}^{u}$. $I_{i,t}^{u} = \sum_{k=1}^{K} I_{i,k,t}^{u}$. The computing delay for tasks in the core k can be expressed as

$$T_{i,k,t}^{u} = \frac{I_{i,k,t}^{u} s}{f_{i,k,t}}, \tag{7}$$

where the CPU requires s cycles to process every individual byte.

Since the delay required to find the energy consumption is the sum of the computing delay of all cores in the CPU, the delay consumption is given by

$$T_{i,t}^{comp} = \sum_{k=1}^{K} T_{i,k,t}^{u}. \tag{8}$$

Our research ignores the edge server's energy consumption and only considers that of the device [12]. As the edge server does not use any energy from the terminal device for its task computing, the base station is the only source of electricity for the edge server. The power of a task during core k calculation can be expressed as

$$p_{i,t}^{u} = \kappa_u (f_{i,k,t})^3, \tag{9}$$

where the CPU's effective capacitance parameter is denoted by κ_u. The sum of the computing power of all cores of the CPU is

$$p_{i,k,t}^{comp} = \sum_{k=1}^{K} p_{i,k,t}^{u}. \tag{10}$$

The energy consumption for task computing can be expressed as

$$E_{i,t}^{comp} = p_{i,t}^{comp} T_{i,t}^{comp}. \tag{11}$$

3.3. Delay Model

Since the tasks offloaded to multiple edge servers and the tasks calculated on the local terminal device are run simultaneously during the computing process, the delay in running tasks in the current epoch is determined by the higher of computing delay $T_{i,t}^{c}$ and the transmission delay $T_{i,j,t}^{trans}$. The delay can be expressed as

$$T_{i,t} = \max\left\{T_{i,t}^{c}, T_{i,1,t}^{trans}, T_{i,2,t}^{trans}, \ldots, T_{i,N,t}^{trans}\right\}, \tag{12}$$

where the computing delay $T_{i,t}^{c}$ of the local terminal device task is determined by the highest computing time among the K CPU cores of the device. The computing delay is given by

$$T_{i,t}^{c} = \max\{T_{i,1,t}^{u}, T_{i,2,t}^{u}, \ldots, T_{i,K,t}^{u}\}. \tag{13}$$

3.4. Energy Consumption Model

From the transmission model, the total energy consumed during the transmission process in task execution can be expressed as

$$E_{i,t}^{t} = \sum_{j=1}^{N} E_{i,j,t}^{trans}. \tag{14}$$

The system's energy consumption is indicated by

$$E_{i,t}^{sum} = E_{i,t}^{c} + E_{i,t}^{t}. \tag{15}$$

3.5. Problem Formulation

We optimize the energy consumption of both terminal devices and edge servers to lower the overall energy use of the system, which involves making decisions about offloading, communication resource allocation, and computing resource allocation. Specifically, the optimization problem can be expressed as

$$\begin{aligned} &\min_{w,p,O,f} E = \sum_{i=1}^{M} \sum_{t=1}^{\mathcal{T}} E_{i,t}^{sum}, \\ &s.t. \quad \mathrm{C1}: O_{i,j,t} \in \{0,1\}, \forall i \in M, j \in N, t \in \mathcal{T}, \\ &\qquad \mathrm{C2}: 0 \le T_{i,t} \le T_{t}^{\max}, \forall i \in M, t \in \mathcal{T} \\ &\qquad \mathrm{C3}: p_{\min} \le p_{i,j,t} \le p_{\max}, \forall i \in M, t \in \mathcal{T}, \\ &\qquad \mathrm{C4}: w^{\min} \le w_{i,j,t} \le w^{\max}, \forall i \in M, t \in \mathcal{T}, \\ &\qquad \mathrm{C5}: f^{\min} \le f_{i,k,t} \le f^{\max}, \forall i \in M, t \in \mathcal{T}, k \in K, \end{aligned} \tag{16}$$

where C1 denotes range of offloading ratio, C2 denotes that each task must be completed within the maximum delay, C3 denotes range of transmission power, C4 denotes range of the subcarriers, and C5 denotes range of the CPU computing frequency of the terminal device.

4. Industrial Federated Deep Deterministic Policy Gradient

In this section, we mainly elaborate and introduce the algorithm we invented to solve the optimization problem (16). In the proposed multiedge-terminal collaborative computing, multiple variables need to be considered, such as offloading decisions, communication resource allocation, and computing resource allocation. The proposed optimization is complex and nonconvex, and traditional optimization algorithms have difficulty in directly obtaining analytical solutions. One effective approach is to use DRL. However, traditional DRL is prone to difficulties in convergence or local convergence problems when dealing with complex nonconvex problems. In addition, in the environment of multiedge-terminal Industrial Internet intelligent factories, terminal devices are different. When using the traditional DDPG algorithm to optimize a single terminal device and sum the results, it may lead to suboptimal energy optimization due to differences in convergence accuracy and speed among multiple terminal devices. FL is advantageous in reducing convergence problems, improving algorithm performance, and ensuring data privacy, making it more suitable for the industrial internet scenario. In response to the above problems, we introduce the Industrial Federated Deep Deterministic Policy Gradient (IF-DDPG) algorithm, which is designed for application in the environment of Industrial Internet intelligent factories. This algorithm is built upon the foundations of both DDPG and FL. The IF-DDPG algorithm combined with FL involves multiple DDPG algorithms, and each DDPG algorithm corresponds to a terminal device. The IF-DDPG algorithm enhances the accuracy of both individual DDPG algorithms and the overall accuracy when multiple DDPG algorithms are combined. This is achieved by exchanging model parameters among the DDPG algorithms, allowing for simultaneous optimization of energy consumption for each terminal device and the entire system. Furthermore, factories have high requirements for data privacy.

Since FL does not require offloading local data, data privacy can also be protected by the DDPG algorithm combined with FL [47].

The IF-DDPG algorithm consists of the DDPG and the FL. The DDPG algorithm is run on each edge server, and then federated averaring is applied the neural network parameters from the DDPG algorithm of multiple edge servers to implement the IF-DDPG algorithm. The agent is the main entity in IF-DDPG. The environment refers to the entity that the agent interacts with during the learning process. The mathematical foundation and modeling tool for the algorithm is the MDP [48]. The MDP usually consists of state s, action a, and reward r.

At each epoch, the environment has a state s. It is the summary of the current environment. The state is the basis for algorithmic decisions. Action a is the decision made by the agent due to the present state. The reward r refers to a value that the environment returns to the agent after the agent performs an action. It is usually assumed that the reward is a function of the current state s, current action a, and next epoch state s'. Denote the reward function as $r(s, a, s')$. The algorithm provides specific definitions for states, actions, and rewards, as outlined below:

1. State Space: The state space is denoted by a total of M mobile smart Industrial Internet of Things sensors/terminal devices and N small base stations equipped with edge servers. The system state $s_{i,t}$ of the terminal device i in the current epoch t can be defined as
$$s_{i,t} = (T_t^{\max}, \mathcal{P}_i, \mathcal{P}_j, I_{i,r}^a, I_{i,t}^a), \tag{17}$$
where $T_t^{\max}$ is the maximum delay threshold for completing the task, $\mathcal{P}_i^u$ is the location information of the mobile terminal device, $\mathcal{P}_j^u$ are the location information of the mobile edge servers, specifically represented by $\mathcal{P}_j^u = \{P_1, P_2, \ldots, P_N\}$, $I_{i,r}^a$ is the remaining data size of the task chain at the current epoch t, and $I_{i,t}^a$ denotes the magnitude of task data that must be computed in the current epoch t.
2. Action Space: In our scenario, the agent selects actions based on the environment observed by the MEC system and its current state. The actions include task offloading ratio, number of subcarriers, subcarrier transmission power, and computing frequency of the CPU in mobile smart terminal devices. The action $a_{i,t}$ of the terminal device i in the current epoch t can be expressed as
$$a_{i,t} = (\mathbf{O}_{i,t}, \mathbf{W}_{i,t}, \mathbf{p}_{i,,t}^{Up}, \mathbf{f}_{i,t}^u), \tag{18}$$
where $\mathbf{O}_{i,t}$ is a vector of offloading ratios for offloading to edge servers, which can be specifically represented as $\mathbf{O}_{i,t} = \{O_{i,1,t}, O_{i,2,t}, \ldots, O_{i,N,t}\}$, corresponding to different edge servers. $\mathbf{W}_{i,t}$ is the vector of subcarrier numbers selected by the terminal device for offloading data, which can be specifically represented as $\mathbf{W}_{i,t} = \{w_{i,1,t}, w_{i,2,t}, \ldots, w_{i,N,t}\}$, corresponding to different edge servers. $\mathbf{p}_{i,,t}^{Up}$ is the vector of transmission powers used for offloading to the edge server, which can be specifically represented as $\mathbf{p}_{i,,t}^{Up} = \{p_{i,1,t}^{Up}, p_{i,2,t}^{Up}, \ldots, p_{i,N,t}^{Up}\}$, corresponding to different edge servers. The computational frequency of the device CPU is continuously adjusted. $\mathbf{f}_{i,t}^u$ is the computing frequencies vector of the CPU, which can be specifically represented as $\mathbf{f}_{i,t}^u = \{f_{i,1,t}^u, f_{i,2,t}^u, \ldots, f_{i,K,t}^u\}$, corresponding to different cores of the CPU. The algorithm optimizes the four aforementioned actions together to reduce the entire cost of the system.
3. Reward Function: The actions selected by an agent are influenced by rewards. The agent chooses an action $a_{i,t}$ based on state $s_{i,t}$ and then receives an immediate reward $r_{i,t}$. The algorithm's performance greatly depends on selecting the suitable reward function. The primary aim of the optimization objective in Equation (13) is to decrease the entire energy used by terminal devices, while the reward function's

objective is to maximize the reward. As a result, energy consumption should be inversely associated with the reward function. The reward function is given by

$$r_{i,t} = r_{i,t}(s_{i,t}, a_{i,t}) = -C \cdot E_{i,t}^{sum}, \tag{19}$$

where the reward parameter C is a constant greater than zero. Its purpose is to enhance the effect of the reward and to highlight the quality of the current action more clearly. The DDPG algorithm can be used to find the action that maximizes the value function q and achieve lower energy consumption for individual terminal devices.

The policy π maps each state to a corresponding action. Given a particular state, the policy defines a probability distribution over the actions that the agent may take:

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s). \tag{20}$$

The policy is stationary and does not change over time. As a result, energy consumption should be inversely associated with the reward function. It can be expressed as

$$U_t = \sum_{\iota=t}^{n} \gamma^{\iota-t} R_\iota, \tag{21}$$

where the discount factor $\gamma \in [0, 1]$, and R_ι represents the reward for the epoch ι that has not yet been observed.

The action value function in MDP calculates the expected sum of rewards that will be obtained by taking a particular action a in the state s. It represents the value of the action. The action value function is given by

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[U_t|S_t = s_t, A_t = a_t]. \tag{22}$$

According to the Bellman equation, the recurrence relationship of the action value function can be expressed as

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})|S_t = s_t, A_t = a_t]. \tag{23}$$

The significance of the action value function is that it can evaluate the future total value after taking action A_t in the state S_t. Therefore, the algorithm finds the best policy by choosing the action that yields the highest value of Q. Thus, the optimal action value function $Q_*(s_t, a_t)$ finds the best course of action in each state. The optimal action value function is given by

$$Q_*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t). \tag{24}$$

The optimal policy is given by

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max\limits_{a_t \in \mathcal{A}} Q_*(s_t, a_t) \\ 0 & \text{otherwise} \end{cases}. \tag{25}$$

If Q_* can be known, the action $a_* = \arg\max\limits_{a_t \in \mathcal{A}} Q_*(s_t, a_t)$ that should be performed by the state S_t at any epoch can be obtained. However, the functional expression of Q_* is unknown in reinforcement learning; therefore, it is necessary to approximate Q_*. One of the most common methods is the DQN algorithm [34]. It has a deep neural network $Q(s_t, a_t; \mathbf{w}_t)$ to approximate $Q_*(s_t, a_t)$ and is trained using the temporal difference method. By using deep neural networks, the DQN algorithm solves problems with large state and action spaces. However, the DQN algorithm has low efficiency and is unable to handle continuous action space problems. An effective way to apply DRL methods such as DQN to continuous domains is to discretize the action space, but this introduces a significant problem, known as the curse of dimensionality: training becomes arduous due to the exponential rise in the number of actions that must be considered with each added degree of freedom.

Furthermore, simply discretizing the action space will remove information about the structure of the action domain.

The IF-DDPG algorithm is a DRL algorithm based on policy gradients. It consists of a policy network and a value network. The policy network controls the agent to take actions, and it makes actions a based on the state s. The value network does not directly control the agent but instead provides a score for action a based on the state s to guide the policy network to make improvements. In addition, IF-DDPG enhances the stability and robustness of the algorithm by adopting concepts from DQN, such as experience replay and a separate target network, to minimize data correlation

As shown in Figure 2, the policy network is called the Actor and is a deterministic function denoted by the function $a = \pi(s; \mathbf{w}_a)$, where $\mathbf{w}_a$ corresponds to the policy network's parameters. The input of the policy network is the state s, and the output is a specific deterministic action a. Based on the state s, the value network evaluates the quality of the action a, which is also the output of the value network. The IF-DDPG algorithm requires training two neural networks so that they improve together. Enhancing the accuracy of the value network's scoring and the decision-making performance of the policy network is the desired outcome.

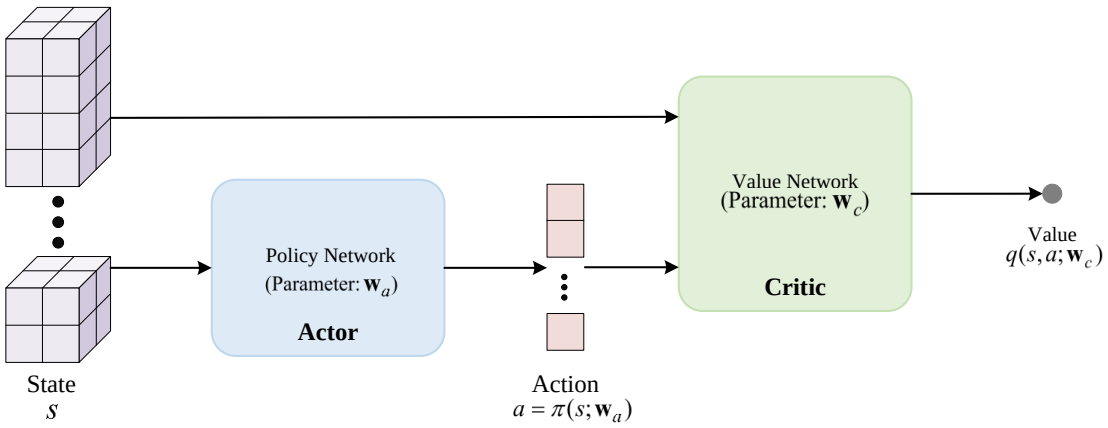

Figure 2. Core of IF-DDPG.

The IF-DDPG algorithm is shown in Figure 3. First, it is necessary to update the policy network to make better decisions. Learning the policy network requires the deterministic policy gradient. The policy network calculates action a based on the input state s to control the movement of the agent, and thence, the policy network is also called an actor. Training the policy network requires the help of the value network. The value network $q(s, a; \mathbf{w}_c)$ evaluates the quality of the action a to guide the policy network to make improvements. The policy network is not equipped to determine the quality of an action, as only the value network provides the evaluation. A higher output value from the value network indicates a better action. Therefore, the goal is to improve the parameters of the policy network $\mathbf{w}_a$ so that the output of the value network $q(s, a; \mathbf{w}_c)$ is larger.

The input of a value network is the state s_t and action a_t at the epoch t. The action a_t is computed by adding Gaussian noise to the output of the policy network $\pi(s_t; \mathbf{w}_a)$. The reason for adding Gaussian noise is to improve policy exploration and increase the randomness of action selection. The first step in iterative algorithms is to initialize the state s_t. For a given state s_t, the policy network will output a deterministic action a_t. Each iteration inputs s_t and a_t into the environment to obtain r_t, initializing the state s_{t+1} and then obtaining a resource transition s_t, a_t, r_t, s_{t+1}. Experience replay is utilized to store resource transitions in the experience replay buffer. When the network is updated, when the buffer reaches full capacity, the algorithm extracts resource groups from the buffer for training purposes. For a fixed-input state of the value network, the only factor that affects the value is the parameter $\mathbf{w}_a$ of the policy network. To accomplish the objective of

increasing the value network's output, we make use of the deterministic policy gradient for updating the policy network. The method requires computing the gradient of $q(s_t, a_t; \mathbf{w}_c)$ with respect to $\mathbf{w}_a$ and then using gradient ascent to update $\mathbf{w}_a$. This gradient is called the policy gradient $\mathbf{g}_t$. The $\mathbf{g}_t$ can be expressed as

$$\mathbf{g}_t = \frac{\partial q[s_t, \pi(s_t; \mathbf{w}_a); \mathbf{w}_c]}{\partial \mathbf{w}_a} = \frac{\partial a_t}{\partial \mathbf{w}_a} \cdot \frac{\partial q(s_t, a_t; \mathbf{w}_c)}{\partial a_t}. \tag{26}$$

This is the gradient of the value q with respect to the parameters $\mathbf{w}_a$ of the policy network. The gradient could be calculated using the chain rule. This is equivalent to passing the gradient from the value q to the action a_t and then from a_t to the policy network. Finally, use gradient ascent to renew the parameters of the policy network $\mathbf{w}_a$, which is given by

$$\mathbf{w}_a \leftarrow \mathbf{w}_a + \beta \cdot \mathbf{g}_t, \tag{27}$$

where β is the parameter that determines the speed of learning for the policy network. Updating $\mathbf{w}_a$ allows the value q to become larger, i.e., the value network believes the policy will become better.

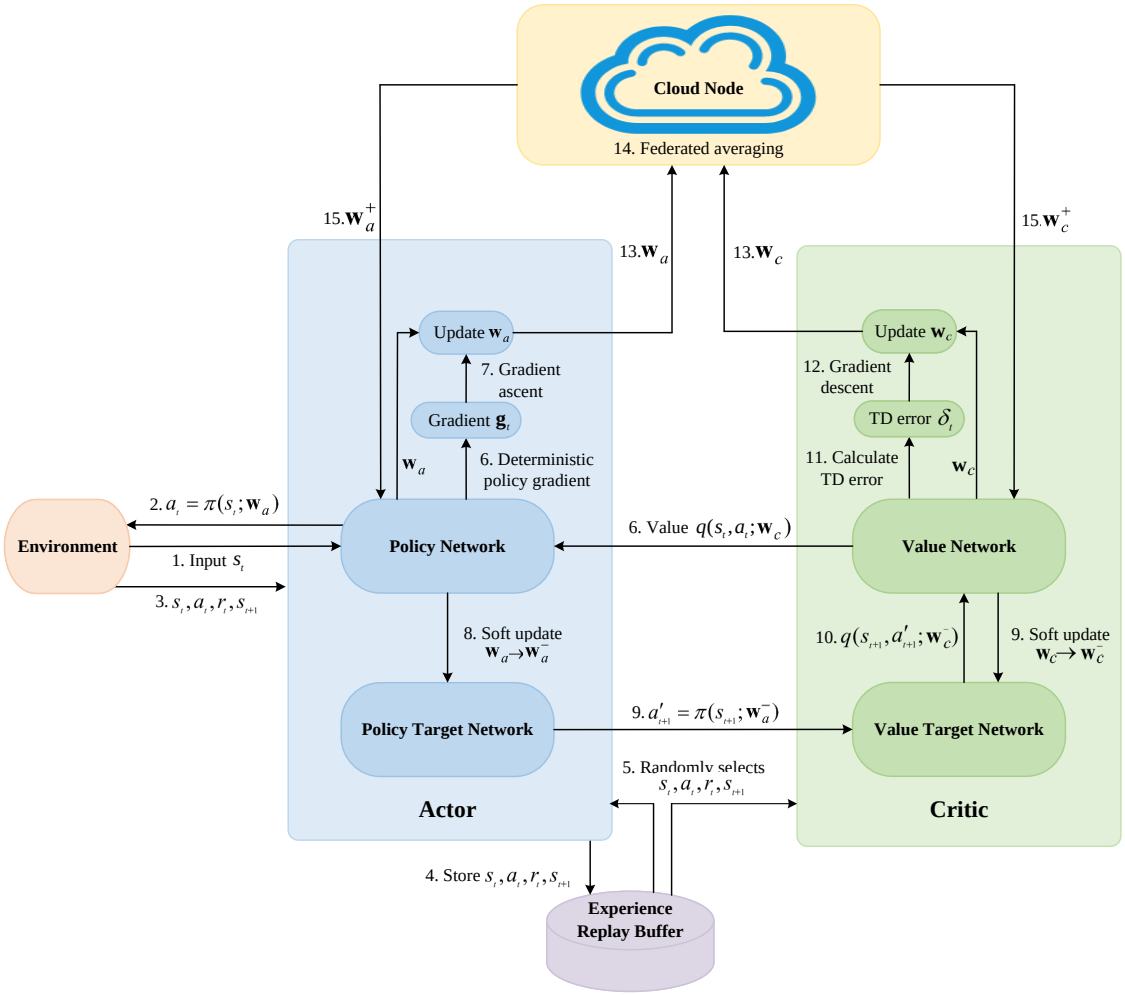

Figure 3. Diagram of IF-DDPG.

After updating the policy network to improve decision making, the subsequent step is to update the value network to make its evaluation more accurate. The value network is also known as a critic. First, the value network is updated using the temporal difference (TD).

The purpose of TD is to make the results on both sides of the equation as close as possible. The equation for the TD can be expressed as

$$q(s_t, a_t; \mathbf{w}_c) \approx r_t + \gamma q(s_{t+1}, a_{t+1}; \mathbf{w}_c). \tag{28}$$

The value network predicts the action value at epoch t, which is denoted by $q_t = q(s_t, a_t; \mathbf{w}_c)$. Next, the value network will predict the action value at the epoch $t+1$ by incorporating the soft update using the target network. Given the state s_{t+1} at epoch $t+1$, the input s_{t+1} is fed into the policy network π combined with the target network, which then computes the next action a'_{t+1}, denoted $a'_{t+1} = \pi(s_{t+1}; \mathbf{w}_a^-)$. The action a'_{t+1} is not the action that the agent performs; it only updates the value network. Then, s_{t+1} and a'_{t+1} are entered into the value network combined with the target network to compute the action value at epoch $t+1$ and obtain $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$. Calculating the TD error is the subsequent step. The TD error is denoted as δ_t

$$\delta_t = q_t - (r_t + \gamma \cdot q_{t+1}), \tag{29}$$

where $r_t + \gamma \cdot q_{t+1}$ is the TD target, consisting of a portion that is the observed reward r_t and another portion that is the value network's prediction q_{t+1}, where γ denotes the discount factor. The TD target is closer to the real value than a simple prediction of q_t, which encourages q_t to approach the TD target to decrease the TD error. Finally, the algorithm performs a gradient descent to renew the parameters $\mathbf{w}_c$ of the value network, which is given by

$$\mathbf{w}_c \leftarrow \mathbf{w}_c - \theta \cdot \delta_t \cdot \frac{\partial q(s_t, a_t; \mathbf{w}_c)}{\partial \mathbf{w}_c}, \tag{30}$$

where $\frac{\partial q(s_t, a_t; \mathbf{w}_c)}{\partial \mathbf{w}_c}$ is the gradient, and θ is the network learning rate. By performing gradient descent, we can reduce the TD error and close the difference between the forecasts of the value network and the TD target. The convergence analysis of the gradient update is shown in Appendix A.

Inspired by FL, in the Industrial Internet system composed of multiple devices, the network parameters $\mathbf{w}_{a,i}$ and $\mathbf{w}_{c,i}$ corresponding to the terminal device i can be uploaded to the cloud node connected to the edge server. The network parameters $\mathbf{w}_{a,i}$ and $\mathbf{w}_i$ are called local models. The cloud node processes the uploaded local model using federated averaging to obtain the global model:

$$\mathbf{w}_a^+ = \frac{1}{M} \sum_{i=1}^{M} \mathbf{w}_{a,i}, \tag{31}$$

$$\mathbf{w}_c^+ = \frac{1}{M} \sum_{i=1}^{M} \mathbf{w}_{c,i}, \tag{32}$$

where $\mathbf{w}_a^+$ and $\mathbf{w}_c^+$ are the policy network parameters and value network parameters after the federation average, M is the number of terminal devices in the system, and each terminal device corresponds to a DDPG algorithm. Next, the cloud node transmits the network parameters of the global model, i.e., $\mathbf{w}_a^+$ and $\mathbf{w}_c^+$, to the policy and value networks of each DDPG algorithm running on the edge server, replacing the original network parameters $\mathbf{w}_a$ and $\mathbf{w}_c$.

At the same time, target networks also need to update their parameters. Set a hyperparameter τ and $\tau \in (0, 1)$. Take the weighted average of $\mathbf{w}_c$ and $\mathbf{w}_c^-$ to obtain a new $\mathbf{w}_c^-$:

$$\mathbf{w}_c^- \leftarrow \tau \cdot \mathbf{w}_c + (1 - \tau) \cdot \mathbf{w}_c^-. \tag{33}$$

Similarly, take the weighted average of $\mathbf{w}_a$ and $\mathbf{w}_a^-$ to obtain a new $\mathbf{w}_a^-$:

$$\mathbf{w}_a^- \leftarrow \tau \cdot \mathbf{w}_a + (1 - \tau) \cdot \mathbf{w}_a^-. \tag{34}$$

In the end, after the algorithm obtains better value network parameters $\mathbf{w}_c$ and policy network parameters $\mathbf{w}_a$, it outputs better actions a_{t+1} in the next iteration. At the conclusion of the current iteration, the algorithm assigns the current state s_{t+1} to s_t to complete the closed loop of the algorithm. Algorithm 1 shows the IF-DDPG algorithm.

Algorithm 1 IF-DDPG

Initialize the state s_t
if the specified number of epochs has not been completed **then**
  Input the state s_t into the policy network
  The policy network outputs the noise-added determinate action $a_t = \pi(s_t; \mathbf{w}_a)$
  Input s_t and a_t into the environment results in r_t
  Initialize the state s_{t+1}
  Get a resource transition: s_t, a_t, r_t, s_{t+1} and store it in the experience replay buffer
  if the buffer capacity is full **then**
    Extract a resource transition s_t, a_t, r_t, s_{t+1} from the buffer
    Use the deterministic policy gradient to calculate the gradient $\mathbf{g}_t = \frac{\partial q[s_t, \pi(s_t; \mathbf{w}_a); \mathbf{w}_c]}{\partial \mathbf{w}_a} = \frac{\partial a_t}{\partial \mathbf{w}_a} \cdot \frac{\partial q(s_t, a_t; \mathbf{w}_c)}{\partial a_t}$ of q with respect to $\mathbf{w}_a$
    Use gradient ascent to update the policy network parameters $\mathbf{w}_a \leftarrow \mathbf{w}_a + \beta \cdot \mathbf{g}_t$
    The value network predicts the action value $q(s_t, a_t; \mathbf{w}_c)$ at the epoch t
    Input s_{t+1} into the policy network combined with the target network π and compute the next action $a'_{t+1} = \pi(s_{t+1}; \mathbf{w}_a^-)$
    Input s_{t+1} and a'_{t+1} into the value network combined with the target network to get $q_{t+1} = q(s_{t+1}, a'_{t+1}; \mathbf{w}_c^-)$
    Calculate TD error: $\delta_t = q_t - (r_t + \gamma \cdot q_{t+1})$
    Use gradient descent to update the value network parameters $\mathbf{w}_c \leftarrow \mathbf{w}_c - \theta \cdot \delta_t \cdot \frac{\partial q(s_t, a_t; \mathbf{w}_c)}{\partial \mathbf{w}_c}$
    Upload the network parameters $\mathbf{w}_{a,i}$ and $\mathbf{w}_{c,i}$ of the terminal device i to the cloud node
    Federated average $\mathbf{w}_a^+ = \frac{1}{M} \sum_{i=1}^{M} \mathbf{w}_{a,i}$ and $\mathbf{w}_c^+ = \frac{1}{M} \sum_{i=1}^{M} \mathbf{w}_{c,i}$ in cloud node
    Transmits $\mathbf{w}_a^+$ and $\mathbf{w}_c^+$ to the edge servers to replace the original network parameters $\mathbf{w}_a$ and $\mathbf{w}_c$
  end if
  Assign the current state s_{t+1} to s_t
end if

5. Results and Analysis

This section illustrates the proposed multiedge-terminal collaborative industrial mobile edge computing solution based on IF-DDPG in the Industrial Internet intelligent factories scenario through simulation results. We validate IF-DDPG's performance in different situations and compare it with other algorithms.

5.1. Simulation Settings

Table 1 displays the parameters for the simulation environment. In the scenario of an Industrial Internet intelligent factory, assume a flat square area of size 100×100 m^2 with N small base stations of edge servers and M mobile smart industrial terminal devices randomly distributed within the area. The transmission bandwidth B of the subcarrier denotes 1 MHz. The channel gain at a reference distance of one meter is $\alpha_0 = -50$ dB, and the system noise power of the receiver is $\delta^2 = -100$ dBm. The minimum number of subcarriers is $w^{\min} = 1$, and the maximum number of subcarriers is $w^{\max} = 10$. The minimum transmission power $p_{\min}$ of the subcarrier is 1 w, and the maximum transmission power $p_{\max}$ of the subcarrier is 10 w. The total data size of the task chain I is 2 Mbits, and the value range of the task data volume I^a is randomly distributed in [60–80] Kbits. We set the minimum CPU clock frequency for the terminal device $f^{\min}$ is 0.1 GHz, and the maximum clock frequency $f^{\max}$ is 0.2 GHz. The quantity of CPU cycles needed to process one-bit

data s is denoted by 1000 cycles/bit. The effective capacitance parameter is $\kappa_u = 10^{-27}$ [49]. Finally, we set the $T_t^{\max}$ is 100 ms [50].

In addition, in the IF-DDPG algorithm, the experimental setting discount factor γ is 0.001, the τ of the target network is 0.01, and the reward parameter C is 100. The experiment uses TensorFlow 2 as the learning framework of IF-DDPG on a PC (CPU: AMD Ryzen 7 6800H, 16 GB RAM).

Table 1. Simulation Parameters.

Parameter	Value	Parameter	Value
Area	100×100 m ^{2}	I	2 Mbits
B	1 MHz	I^a	[60–80] Kbits
α_0	-50 dB	$f^{\min}$	0.1 GHz
δ^2	-100 dBm	$f^{\max}$	0.2 GHz
$w^{\min}$	1	s	1000 cycles/bit
$w^{\max}$	10	κ_u	10^{-27}
$p_{\min}$	1 w	γ	0.001
$p_{\max}$	10 w	τ	0.01
$T_t^{\max}$	100 ms	C	100

The four algorithms compared in this paper are as follows:

- DQN-based edge computing offloading and resource allocation algorithm: A comparison between the DQN based on discrete action space and IF-DDPG [51].
- AC-based edge computing offloading and resource allocation algorithm: A comparison between AC based on continuous action space and IF-DDPG [52].
- Dueling DQN-based edge computing offloading and resource allocation algorithm (DDQN): DDQN, in contrast to DQN, stores the experience sample data from the agent's interactions with the environment in an experience pool. A tiny batch of data from the experience pool is chosen at random for training [13].
- DDPG-based edge computing offloading and resource allocation algorithm: A comparison between the traditional DDPG without incorporating FL and IF-DDPG. The above comparison algorithms are similar to IF-DDPG, which are all DRL algorithms.

5.2. Simulation Results

5.2.1. Parameter Analysis

Firstly, before comparing and validating the performance of the IF-DDPG algorithm, we should set two important deep neural network hyperparameters in the algorithm: the learning rate β for the policy network and the learning rate θ for the value network. The experimental environment consists of 4 edge servers and 4 terminal devices, where the quantity of available CPU cores for terminal devices is $K = 4$. Table 2 shows 5 learning rate allocation schemes for the experimental settings, which are Scheme 1: $\beta = 0.01$ & $\theta = 0.02$, Scheme 2: $\beta = 0.001$ & $\theta = 0.002$, Scheme 3: $\beta = 0.0001$ & $\theta = 0.0002$, Scheme 4: $\beta = 0.00001$ & $\theta = 0.00002$, and Scheme 5: $\beta = 0.000001$ & $\theta = 0.000002$. Figures 4 and 5 show the convergence performance of 5 learning rate allocation schemes for the IF-DDPG algorithm in terms of reward values and total system energy consumption after 1500 iterations. From the figures, it can be observed that when Scheme 1 is employed, the IF-DDPG algorithm converges. However, it is observed that in the middle and later stages, the curve tends to flatten and converge to a local optimal solution. The reason for this is that a higher learning rate leads to larger update steps for the deep neural network. It is more prone to local convergence. When Scheme 5 is employed, the curve oscillates repeatedly and changes slowly, indicating that the IF-DDPG algorithm fails to converge currently.

The reason is that the smaller the learning rate, the slower the update rate of the deep neural network, and more iterations are needed for possible convergence. When Scheme 3 is employed, the curve changes quickly and the magnitude of change is greater compared with the comparison algorithm. This indicates that at this time, the IF-DDPG algorithm converges and has the best convergence performance. Therefore, the learning rate setting of Scheme 3 will be used consistently throughout the subsequent experimental process.

Table 2. Schemes settings.

Schemes	Learning Rate β	Learning Rate θ
1	0.01	0.02
2	0.001	0.002
3	0.0001	0.0002
4	0.00001	0.00002
5	0.000001	0.000002

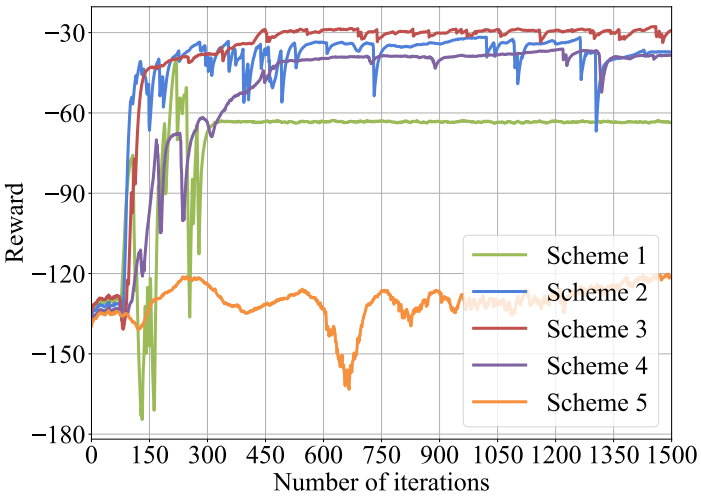

Figure 4. Convergence of learning rates with reward.

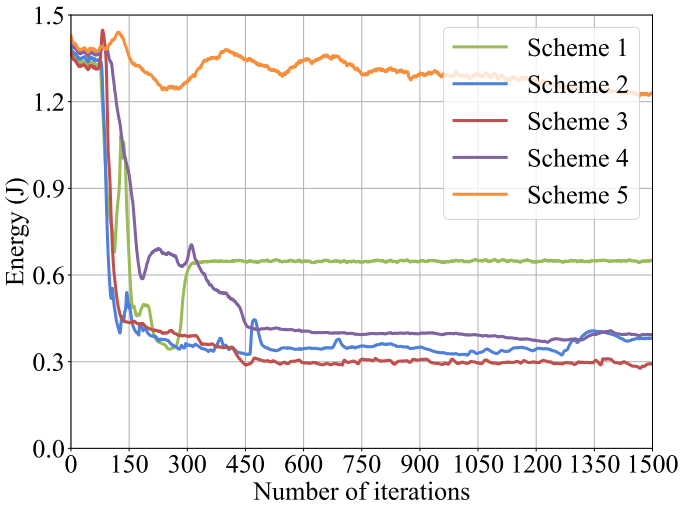

Figure 5. Convergence of learning rates with system energy consumption.

5.2.2. Algorithm Performance Comparison

Next, the experiment will verify the performance of IF-DDPG. The experimental environment consists of 4 edge servers and 4 terminal devices, where the number of available CPU cores for each terminal device is $K = 4$. To verify the performance of IF-DDPG, four algorithms are compared experimentally: DQN, AC, DDQN, and DDPG. Figure 6 shows the convergence performance of IF-DDPG compared with the four other algorithms in terms of reward values after 1500 iterations. From Figure 6, it can be observed that after the convergence of five algorithms, the average reward of IF-DDPG is notably greater than that of the DDPG, the AC, the DQN, and the DDQN. Additionally, the IF-DDPG exhibits lower fluctuations in the later stages of the curve. IF-DDPG's offloading strategy and resource allocation method exhibit clearer advantages over the four comparison algorithms. In addition, as shown in Figure 6, IF-DDPG exhibits a larger curve growth rate and faster convergence speed. The IF-DDPG allows multiple DDPGs to learn from each other through the FL. By sharing their deep learning network parameters, it achieves higher convergence performance and learning efficiency.

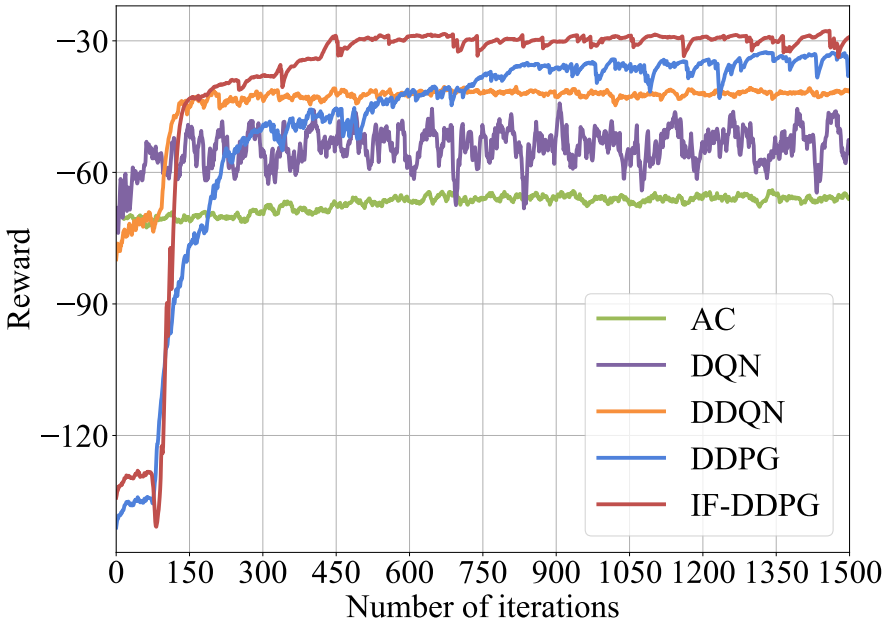

Figure 6. Comparison of algorithms convergence.

After the 1500 iterations, Figure 7 demonstrates the energy consumption optimization performance of IF-DDPG in comparison to the 4 other algorithms. From the figure, it can be observed that after the convergence of the five algorithms, using IF-DDPG to optimize energy consumption results in lower average and minimum energy consumption compared with the comparison algorithms. Additionally, the convergence speed of the curve of IF-DDPG is faster. Compared with the traditional DDPG, the IF-DDPG optimized minimum energy consumption reduced by 15.2%. Compared with the DDQN, it reduced by 31.7%. Compared with the DQN, it reduced by 38.7%. Finally, compared with the AC algorithm, it reduced by 50.5%.

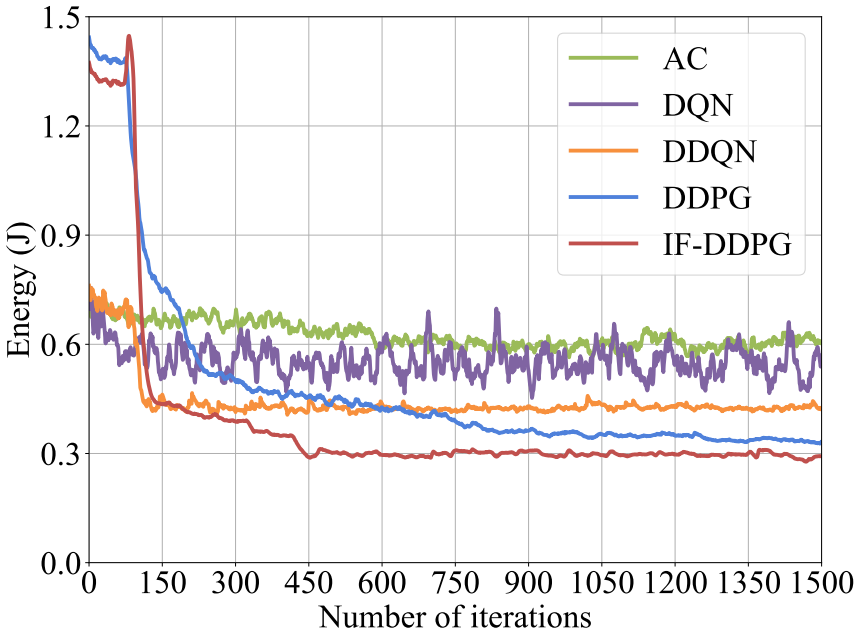

Figure 7. Comparison of algorithms' energy consumption optimization performance.

Table 3 shows the comparison between IF-DDPG and the exhaustion search method for optimizing the minimum value of the total energy consumption of the system. To reduce the complexity of the exhaustion search method, environment simulation validation was performed using two edge servers and four terminal devices. From Table 3, it can be observed that the minimum energy consumption of the exhaustive search method is less than that of IF-DDPG, but the computation time required to obtain the minimum energy consumption using the exhaustive search method is much greater than that of IF-DDPG. The enormous computational time consumption is unacceptable in practical intelligent factory environments. Therefore, although IF-DDPG is slightly inferior to the exhaustive search method in terms of performance, it is easier to implement in practical applications, and thus has greater practical value.

Table 3. Comparison between IF-DDPG and method of exhaustion.

	Energy (J)	**Running Time (s)**
IF-DDPG	0.219	283
Exhaustion	0.178	398,049

5.2.3. Effects of Edge Servers, Terminal Devices, and Number of CPU Cores

Next, the experiment compares the performance of IF-DDPG and the comparison algorithm under various quantities of edge servers and terminal devices in an intelligent factory environment. Figure 8 demonstrates the contrast of the minimum energy consumption of IF-DDPG and the 4 comparison algorithms after 1500 iterations in an environment with different counts of edge servers. Figure 8 shows how the total energy consumption of the system grows as the quantity of edge servers increases. The curve of IF-DDPG is smoother than the comparison algorithms, with relatively lower minimum energy consumption, and it less affected by the quantity of edge servers. Moreover, IF-DDPG's benefit becomes increasingly apparent as the quantity of edge servers grows, because IF-DDPG is an improvement over DDPG. The DDPG utilizes two deep neural networks and incorporates the experience replay mechanism. In comparison wihth the baseline algorithm, the IF-DDPG demonstrates higher efficiency in handling optimization problems with massive state and action spaces. In the scenario involving multiple edge servers, IF-DDPG outperforms DDPG in terms of performance.

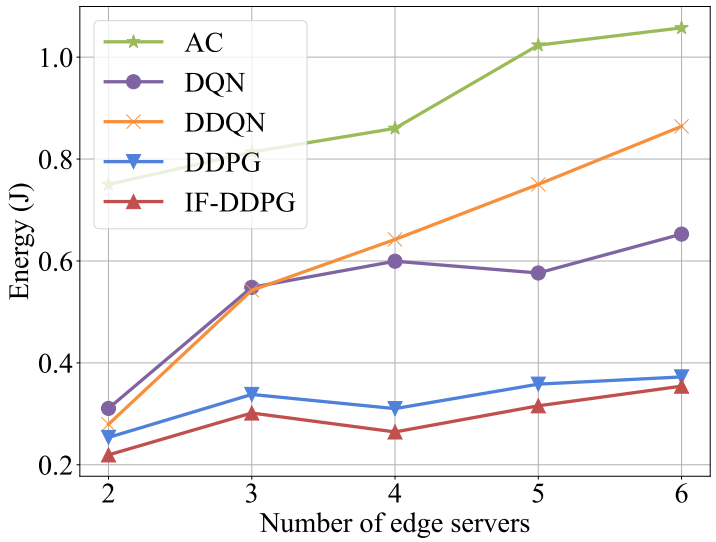

Figure 8. Effect of the number of edge servers on performance.

Figure 9 shows the comparison of the minimum energy consumption of IF-DDPG and the 4 comparison algorithms after 1500 iterations in an environment with different quantities of terminal devices. From Figure 9, it can be observed that as the quantity of terminal devices grows, the total energy consumption of the system also increases. Similar to Figure 8, the curve growth of IF-DDPG is smaller than the contrast algorithm, and the curve rises more smoothly. It has a relatively lower minimum energy consumption and is relatively less affected by the quantity of terminal devices. As the quantity of terminal devices increases, the advantages of IF-DDPG become more obvious. This is because IF-DDPG utilizes FL to enable each device's corresponding DDPG to learn from one another and share their deep learning network parameters, resulting in better learning performance. In contrast, the comparison algorithm for each terminal device learns independently, resulting in lower learning efficiency compared with IF-DDPG.

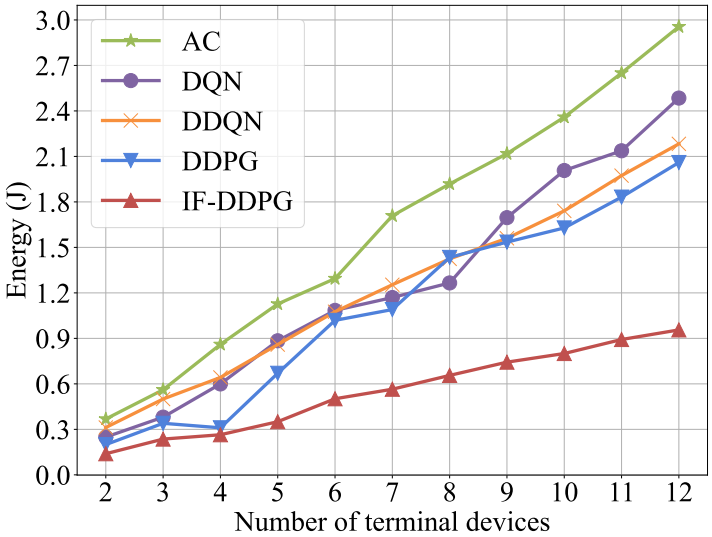

Figure 9. Effect of the number of terminal devices on performance.

Finally, the experiment compared the effect of the number of available CPU cores for terminal devices with different quantities on the performance of IF-DDPG and the contrast algorithm. Figure 10 shows a comparison of the minimum energy consumption between IF-DDPG and the traditional DDPG after 1500 iterations under different numbers of available CPU cores in the environment. Figure 10 observes that the system's overall

energy consumption lowers as the number of available CPU cores rises. Compared with the traditional DDPG, the curve of IF-DDPG has a more obvious downward trend, and has lower minimum energy consumption, because the increase in the number of available cores allows more dimensional resource scheduling. IF-DDPG uses these schedulable resources more effectively to obtain a better resource scheduling strategy, resulting in better optimization of the overall energy consumption of the system.

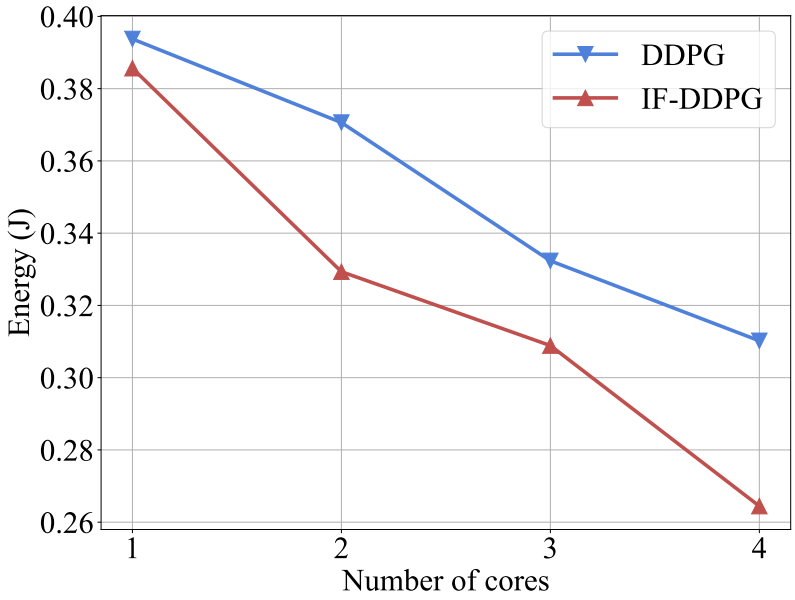

Figure 10. Effect of the number of available CPU cores on performance.

6. Conclusions

In this paper, we research the multiedge-terminal MEC system of an intelligent factory in an Industrial Internet scenario to reduce the total energy consumption of the MEC system through multiedge-terminal computing offloading and resource allocation. The system consists of multiple small base stations connecting to edge servers and multiple mobile smart industrial terminal devices. The terminal devices choose to compute tasks locally or wirelessly offload them to the edge servers. For this system, we designed the IF-DDPG algorithm, which combines FL and DRL. This algorithm achieves a better strategy by jointly optimizing task offloading ratio, subcarrier number, transmission power, and computation frequency, thereby reducing the overall energy consumption of the multiedge-terminal MEC system. Furthermore, the paper details the principle of the IF-DDPG algorithm utilized in this work to tackle the optimization issue. Then, the paper analyzes the effect of variable parameters on IF-DDPG and compares the performance of IF-DDPG with the comparison algorithms under different conditions through simulation analysis. IF-DDPG's convergence was additionally theoretically examined. The simulation results demonstrate that the IF-DDPG algorithm outperforms in optimizing the total energy consumption of the system. In the future, we will explore the simultaneous optimization of system energy consumption and delay in the Industrial Internet multiedge-terminal MEC system to minimize the total system cost.

Author Contributions: Conceptualization, X.L. and J.Z.; methodology, X.L., J.Z. and C.P.; software, X.L. and J.Z.; validation, X.L., J.Z. and C.P.; formal analysis, X.L. and J.Z.; investigation, X.L. and J.Z., and C.P.; resources, X.L.; writing—original draft preparation, X.L. and J.Z.; writing—review and editing, X.L., J.Z. and C.P.; supervision, X.L.; project administration, X.L.; funding acquisition, X.L. and C.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by Beijing Natural Science Foundation L222004, Beijing Natural Science Foundation Haidian Original Innovation Joint Fund (No. L212026), R&D Program of Beijing Municipal Education Commission (KM202211232011).

Institutional Review Board Statement:

Informed Consent Statement:

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

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

Appendix A. Convergence Proof of Gradient Update

Consider the following iterations of gradient updates:

Assumption A1. *The network function $f(a)$ is L-smooth and μ-strongly.*

Assumption A1 guarantees the feasibility of linear regression and the updated rules of deep learning. Therefore, there is the following theorem.

Theorem A1. *If the function $f(a)$ is L-smooth and μ-strongly, then set the step size (learning rate) to $\theta = \frac{1}{L}$, where L is the smoothing coefficient. Therefore, we obtain*

$$f(a_{t+1}) - f(a^*) \leq \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^t \|a_0 - a^*\|^2, \tag{A1}$$

where a^* is the optimal solution, and the gradient update ensures the convergence to the global optimal point.

Proof of Theorem 1. First, due to smoothness and $L > 0$, for the function $f(a)$ for any a and a', we have

$$f(a') \leq f(a) + \nabla f(a)^\top (a' - a) + \frac{L}{2}\|a' - a\|^2. \tag{A2}$$

Substituting a_t and a_{t+1} into the above formula, we obtain

$$\begin{aligned} f(a_{t+1}) &\leq f(a_t) + \nabla f(a_t)^\top (a_{t+1} - a_t) + \frac{L}{2}\|a_{t+1} - a_t\|^2 \\ &= f(a_t) + \nabla f(a_t)^\top (a_t - \theta \nabla f(a_t) - a_t) + \frac{L}{2}\|a_t - \theta \nabla f(a_t) - a_t\|^2 \\ &= f(a_t) - \nabla f(a_t)^\top \theta \nabla f(a_t) + \frac{L}{2}\|\theta \nabla f(a_t)\|^2 \\ &= f(a_t) - \theta \|\nabla f(a_t)\|^2 + \frac{L}{2}\theta^2 \|\nabla f(a_t)\|^2 \\ &= f(a_t) - \left(1 - \frac{L}{2}\theta\right)\theta \|\nabla f(a_t)\|^2 \\ &= f(a_t) - \frac{\|\nabla f(a_t)\|^2}{2L}. \end{aligned} \tag{A3}$$

Let $\Psi_t = f(a_t) - f(a^*)$; the above formula can be expressed as

$$\Psi_{t+1} - \Psi_t \leq -\frac{\|\nabla f(a_t)\|^2}{2L}. \tag{A4}$$

According to strong convexity, we obtain $\mu > 0$. For the function $f(a)$, if for any a' and a, we have

$$f(a') \geq f(a) + \nabla f(a)^\top (a' - a) + \frac{\mu}{2}\|a' - a\|^2. \tag{A5}$$

Then, according to the strong convexity, let $a' = a^*$; we obtain the following:

$$\Psi_t \leq -\nabla f(a_t)^\top (a^* - a_t) - \frac{\mu}{2}\|a_t - a^*\|^2. \tag{A6}$$

According to the triangle inequality, for any a, we have

$$\|a_{t+1} - a\|^2 = \|a_t - a\|^2 - 2\theta \nabla f(a_t)^\top (a_t - a) + \theta^2 \|\nabla f(a_t)\|^2. \tag{A7}$$

Therefore, let $a' = a^*$; we obtain

$$\begin{aligned} \Psi_t &\leq -\nabla f(a_t)^\top (a^* - a_t) - \frac{\mu}{2}\|a_t - a^*\|^2 \\ &\leq \frac{L-\mu}{2}\|a_t - a^*\|^2 - \frac{L}{2}\|a_{t+1} - a^*\|^2 + \frac{\|\nabla f(a_t)\|^2}{2L}. \end{aligned} \tag{A8}$$

Next, add the above formula to (A4) to obtain

$$\Psi_{t+1} \leq \frac{L-\mu}{2}\|a_t - a^*\|^2 - \frac{L}{2}\|a_{t+1} - a^*\|^2. \tag{A9}$$

Rearranging the terms in the above inequality, and then according to $\Psi_{t+1} > 0$, we obtain

$$\|a_{t+1} - a^*\|^2 \leq \left(1 - \frac{\mu}{L}\right)\|a_t - a^*\|^2 \leq \left(1 - \frac{\mu}{L}\right)^{t+1}\|a_0 - a^*\|^2. \tag{A10}$$

Finally, we deduce that

$$\begin{aligned} f(a_{t+1}) - f(a^*) &= \Psi_{t+1} \\ &\leq \frac{L-\mu}{2}\|a_t - a^*\|^2 - \frac{L}{2}\|a_{t+1} - a^*\|^2 \\ &\leq \frac{L-\mu}{2}\|a_t - a^*\|^2 \\ &\leq \frac{L-\mu}{2}\left(1 - \frac{\mu}{L}\right)^t \|a_0 - a^*\|^2 \\ &\leq \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^t \|a_0 - a^*\|^2. \end{aligned} \tag{A11}$$

As L-smooth constrains the upper bound of the function $f(a)$, and μ-strongly constrains the lower bound of the function $f(a)$, we have

$$\frac{\mu}{2}\|a' - a\|^2 \leq f(a') - f(a) - \nabla f(a)^\top (a' - a) \leq \frac{L}{2}\|a' - a\|^2. \tag{A12}$$

Thus, we have $\mu < L$, and consequently obtain $1 - \frac{\mu}{L} < 1$. Take the limit of $t \to \infty$ on the left and right sides of the inequality sign in the above formula to obtain

$$\lim_{t\to\infty}[f(a_t) - f(a^*)] = 0. \tag{A13}$$

Therefore, under the assumption of L-smooth and μ-strongly, gradient updates guarantee convergence to the global optimum. □

References

1. Boyes, H.; Hallaq, B.; Cunningham, J.; Watson, T. The Industrial Internet of Things (IIoT) An Analysis Framework. *Comput. Ind.* **2018**, *101*, 1–12. [CrossRef]
2. Liu, W.; Nair, G.; Li, Y.; Nesic, D.; Vucetic, B.; Poor, H.V. On the Latency, Rate, and Reliability Tradeoff in Wireless Networked Control Systems for IIoT. *IEEE Internet Things J.* **2021**, *8*, 723–733. [CrossRef]

3. Bozorgchenani, A.; Mashhadi, F.; Tarchi, D.; Salinas Monroy, S.A. Multi-Objective Computation Sharing in Energy and Delay Constrained Mobile Edge Computing Environments. *IEEE Trans. Mob. Comput.* **2021**, *20*, 2992–3005. [CrossRef]
4. Qiu, T.; Chi, J.; Zhou, X.; Ning, Z.; Atiquzzaman, M.; Wu, D.O. Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2462–2488. [CrossRef]
5. 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]
6. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
7. Yang, L.; Dai, Z.; Li, K. An Offloading Strategy Based on Cloud and Edge Computing for Industrial Internet. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1666–1673. [CrossRef]
8. Jiang, C.; Fan, T.; Gao, H.; Shi, W.; Liu, L.; Cérin, C.; Wan, J. Energy Aware Edge Computing: A Survey. *Comput. Commun.* **2020**, *151*, 556–580. [CrossRef]
9. Kuang, Z.; Li, L.; Gao, J.; Zhao, L.; Liu, A. Partial Offloading Scheduling and Power Allocation for Mobile Edge Computing Systems. *IEEE Internet Things J.* **2019**, *6*, 6774–6785. [CrossRef]
10. Misra, S.; Mukherjee, A.; Roy, A.; Saurabh, N.; Rahulamathavan, Y.; Rajarajan, M. Blockchain at the Edge: Performance of Resource-Constrained IoT Network. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 174–183. [CrossRef]
11. Sun, L.; Wang, J.; Lin, B. Task Allocation Strategy for MEC-Enabled IIoTs via Bayesian Network Based Evolutionary Computation. *IEEE Trans. Ind. Inform.* **2021**, *17*, 3441–3449. [CrossRef]
12. Xu, H.; Li, Q.; Gao, H.; Xu, X.; Han, Z. Residual Energy Maximization-Based Resource Allocation in Wireless-Powered Edge Computing Industrial IoT. *IEEE Internet Things J.* **2021**, *8*, 17678–17690. [CrossRef]
13. Chu, J.; Pan, C.; Wang, Y.; Yun, X.; Li, X. Edge Computing Resource Allocation Algorithm for NB-IoT Based on Deep Reinforcement Learning. *IEICE Trans. Commun.* **2022**, *106* , 439–447. [CrossRef]
14. Li, Z.; He, Y.; Yu, H.; Kang, J.; Li, X.; Xu, Z.; Niyato, D. Data Heterogeneity-Robust Federated Learning via Group Client Selection in Industrial IoT. *IEEE Internet Things J.* **2022**, *9*, 17844–17857. [CrossRef]
15. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic Policy Gradient Algorithms. In Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 387–395.
16. Konečný, J.; McMahan, H.B.; Ramage, D.; Richtárik, P. Federated Optimization: Distributed Machine Learning for on-Device Intelligence. *arXiv* **2016**, arXiv:1610.02527. [CrossRef]
17. Shu, C.; Zhao, Z.; Han, Y.; Min, G. Dependency-Aware and Latency-Optimal Computation Offloading for Multi-User Edge Computing Networks. In Proceedings of the 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), Boston, MA, USA, 10–13 June 2019; pp. 1–9. [CrossRef]
18. Wang, X.; Wang, J.; Zhang, X.; Chen, X.; Zhou, P. Joint Task Offloading and Payment Determination for Mobile Edge Computing: A Stable Matching Based Approach. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12148–12161. [CrossRef]
19. Tran, T.X.; Pompili, D. Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 856–868. [CrossRef]
20. Zhao, J.; Li, Q.; Gong, Y.; Zhang, K. Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 7944–7956. [CrossRef]
21. Kai, C.; Zhou, H.; Yi, Y.; Huang, W. Collaborative Cloud-Edge-End Task Offloading in Mobile-Edge Computing Networks with Limited Communication Capability. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *7*, 624–634. [CrossRef]
22. Wang, T.; Lu, Y.; Wang, J.; Dai, H.-N.; Zheng, X.; Jia, W. EIHDP: Edge-Intelligent Hierarchical Dynamic Pricing Based on Cloud-Edge-Client Collaboration for IoT Systems. *IEEE Trans. Comput.* **2021**, *70*, 1285–1298. [CrossRef]
23. Yaqoob, M.M.; Khurshid, W.; Liu, L.; Arif, S.Z.; Khan, I.A.; Khalid, O.; Nawaz, R. Adaptive Multi-Cost Routing Protocol to Enhance Lifetime for Wireless Body Area Network. *Comput. Mater. Contin.* **2022**, *72*, 1089–1103. [CrossRef]
24. You, C.; Huang, K.; Chae, H.; Kim, B.-H. Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading. *IEEE Trans. Wirel. Commun.* **2017**, *16*, 1397–1411. [CrossRef]
25. Tan, L.; Kuang, Z.; Zhao, L.; Liu, A. Energy-Efficient Joint Task Offloading and Resource Allocation in OFDMA-Based Collaborative Edge Computing. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 1960–1972. [CrossRef]
26. Chen, Y.; Liu, Z.; Zhang, Y.; Wu, Y.; Chen, X.; Zhao, L. Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2021**, *17*, 4925–4934. [CrossRef]
27. Xiao, L.; Lu, X.; Xu, T.; Wan, X.; Ji, W.; Zhang, Y. Reinforcement Learning-Based Mobile Offloading for Edge Computing Against Jamming and Interference. *IEEE Trans. Commun.* **2020**, *68*, 6114–6126. [CrossRef]
28. Chen, Z.; Wang, X. Decentralized Computation Offloading for Multi-User Mobile Edge Computing: A Deep Reinforcement Learning Approach. *EURASIP J. Wirel. Commun. Netw.* **2020**, *2020*, 188. [CrossRef]
29. 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]
30. Li, M.; Gao, J.; Zhao, L.; Shen, X. Adaptive Computing Scheduling for Edge-Assisted Autonomous Driving. *IEEE Trans. Veh. Technol.* **2021**, *70*, 5318–5331. [CrossRef]

31. Yan, L.; Chen, H.; Tu, Y.; Zhou, X. A Task Offloading Algorithm with Cloud Edge Jointly Load Balance Optimization Based on Deep Reinforcement Learning for Unmanned Surface Vehicles. *IEEE Access.* **2022**, *10*, 16566–16576. [CrossRef]
32. Nath, S.; Wu, J. Deep Reinforcement Learning for Dynamic Computation Offloading and Resource Allocation in Cache-Assisted Mobile Edge Computing Systems. *Intell. Converg. Netw.* **2020**, *1*, 181–198. [CrossRef]
33. AlQerm, I.; Pan, J. Enhanced Online Q-Learning Scheme for Resource Allocation with Maximum Utility and Fairness in Edge-IoT Networks. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 3074–3086. [CrossRef]
34. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-Level Control through Deep Reinforcement Learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
35. 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. [CrossRef]
36. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Aguera y Arcas, B. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
37. Konečný, J.; McMahan, H.B.; Yu, F.X.; Richtarik, P.; Suresh, A.T.; Bacon, D. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv* **2016**, arXiv:1610.05492. [CrossRef]
38. Yaqoob, M.M.; Nazir, M.; Yousafzai, A.; Khan, M.A.; Shaikh, A.A.; Algarni, A.D.; Elmannai, H. Modified Artificial Bee Colony Based Feature Optimized Federated Learning for Heart Disease Diagnosis in Healthcare. *Appl. Sci.* **2022**, *12*, 12080. [CrossRef]
39. Yaqoob, M.M.; Nazir, M.; Khan, M.A.; Qureshi, S.; Al-Rasheed, A. Hybrid Classifier-Based Federated Learning in Health Service Providers for Cardiovascular Disease Prediction. *Appl. Sci.* **2023**, *13*, 1911. [CrossRef]
40. Yu, S.; Chen, X.; Zhou, Z.; Gong, X.; Wu, D. When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multitimescale Resource Management for Multiaccess Edge Computing in 5G Ultradense Network. *IEEE Internet Things J.* **2021**, *8*, 2238–2251. [CrossRef]
41. Wang, H.; Kaplan, Z.; Niu, D.; Li, B. Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 1698–1707. [CrossRef]
42. Khodadadian, S.; Sharma, P.; Joshi, G.; Maguluri, S.T. Federated Reinforcement Learning: Linear Speedup Under Markovian Sampling. In Proceedings of the 39th International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022; pp. 10997–11057. Available online: https://proceedings.mlr.press/v162/khodadadian22a/khodadadian22a.pdf (accessed on 24 February 2023).
43. Tianqing, Z.; Zhou, W.; Ye, D.; Cheng, Z.; Li, J. Resource Allocation in IoT Edge Computing via Concurrent Federated Reinforcement Learning. *IEEE Internet Things J.* **2022**, *9*, 1414–1426. [CrossRef]
44. Luo, S.; Chen, X.; Wu, Q.; Zhou, Z.; Yu, S. HFEL: Joint Edge Association and Resource Allocation for Cost-Efficient Hierarchical Federated Edge Learning. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 6535–6548. [CrossRef]
45. Zhu, Z.; Wan, S.; Fan, P.; Letaief, K.B. Federated Multiagent Actor–Critic Learning for Age Sensitive Mobile-Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 1053–1067. [CrossRef]
46. Liu, L.; Zhang, J.; Song, S.H.; Letaief, K.B. Client-Edge-Cloud Hierarchical Federated Learning. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6. [CrossRef]
47. Elbamby, M.S.; Perfecto, C.; Liu, C.-F.; Park, J.; Samarakoon, S.; Chen, X.; Bennis, M. Wireless Edge Computing with Latency and Reliability Guarantees. *Proc. IEEE* **2019**, *107*, 1717–1737. [CrossRef]
48. Luong, N.C.; Hoang, D.T.; Gong, S.; Niyato, D.; Wang, P.; Liang, Y.-C.; Kim, D.I. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3133–3174. [CrossRef]
49. Huang, P.-Q.; Wang, Y.; Wang, K.; Liu, Z.-Z. A Bilevel Optimization Approach for Joint Offloading Decision and Resource Allocation in Cooperative Mobile Edge Computing. *IEEE Trans. Cybern.* **2020**, *50*, 4228–4241. [CrossRef]
50. White Paper on Edge Computing Network of Industrial Internet in the 5G Era. Available online: http://www.ecconsortium.org/Uploads/file/20201209/1607521755435690.pdf (accessed on 24 February 2023).
51. 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.* **2021**, *9*, 1529–1541. [CrossRef]
52. Liu, K.-H.; Hsu, Y.-H.; Lin, W.-N.; Liao, W. Fine-Grained Offloading for Multi-Access Edge Computing with Actor-Critic Federated Learning. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 29 March–1 April 2021; pp. 1–6. [CrossRef]