Received 5 September 2022, accepted 21 September 2022, date of publication 28 September 2022, date of current version 4 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3210248

RESEARCH ARTICLE

Task Offloading and Resource Allocation for Industrial Internet of Things: A Double-Dueling Deep Q-Network Approach

WEIJUN CHENG[1,2], (Member, IEEE), XIAOSHI LIU[1], XIAOTING WANG[1], AND GAOFENG NIE[2], (Member, IEEE)

[1]School of Information Engineering, Minzu University of China, Beijing 100081, China
[2]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Weijun Cheng (weijuncheng@muc.edu.cn)

This work was supported by the Open Foundation of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, under Grant SKLNST-2022-1-14.

ABSTRACT With the development of 5G technology, Mobile Edge Computing (MEC) has become a promising technology that is widely used in the Industrial Internet of Things (IIoT) and other fields. However, the increase in terminal devices and massive data growth has brought new challenges to MEC systems. How to meet the latency requirements of mobile devices while reducing system costs as much as possible is an urgent problem to be solved. To address this problem, we construct a smart factory model and formulate a mixed-integer nonlinear programming problem with the goal of minimizing the weighted sum of task delay and energy consumption. Considering that this problem is non-deterministic polynomial hard (NP-hard), we choose the Deep Q-network (DQN) approach to solve the objective function. In order to avoid the inaccurate Q-value estimation problem of the Double DQN algorithm and the overestimation problem of the Dueling DQN algorithm, we combine them to propose a Double-Dueling DQN (D3QN) algorithm. The simulation results show that the D3QN algorithm significantly outperforms the DQN, the Double DQN, and the Dueling DQN algorithm in reducing the total system cost.

INDEX TERMS Mobile edge computing (MEC), task offloading, deep reinforcement learning, Industrial Internet of Things (IIoT).

I. INTRODUCTION

Driven by the Fourth Industrial Revolution wave, the world has entered an era of intelligence. The traditional interconnection between people and people or people and things has been unable to meet the needs of today's social production. Thanks to the emergence and development of the Internet of Things (IoT) technology, people have broken the barrier of connectivity between things, which makes it possible to realize the Internet of Everything (IoE). Based on this, Germany first proposed the idea of Industry 4.0 at HANNOVER MESSE in 2013 [1]. With the further development of science and technology, the outline of smart manufacturing has gradually become clear. All major industrial countries have also gradually transformed and upgraded traditional manufacturing factories into smart factories [2]. In recent years, on the one hand, smart factories have greatly improved the production efficiency of factories and reduced production costs. Still, on the other hand, their shortcomings have gradually been exposed. The Cloud Computing (CC) centralized processing mode in traditional smart factories has risks such as high delay, high energy consumption, and low security, and it cannot meet some tasks with high requirements such as delay [3]. Furthermore, with the continuous increase in the number of smart devices, the amount of task data will increase exponentially, which will also cause tremendous pressure on the transmission bandwidth of CC.

Fortunately, the emergence of Mobile Edge Computing (MEC) provides the possibility to solve the above problems. MEC is a data offloading solution that improves the

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

computing and storage capabilities on the edge side of the network by deploying edge servers with computing and storage capabilities near mobile terminals. MEC inherits part of the caching and computing power of CC. Because edge computing servers are usually deployed at the edge of the network and are closer to users, they not only make up for the limited computing capability of smart terminal devices but also solve the problems of CC transmission delay and high energy consumption. The development of MEC has greatly improved users' Quality of Experience (QoE) and Quality of Service (QoS) [4]. It can be said that, as a supplement to CC, MEC adds distributed capabilities to traditional cloud centers; meanwhile, the coverage of CC is further extended.

In the current Industrial Internet of Things (IIoT), the main problems of MEC are offloading decisions and resource allocation [5]. Typically, the offloading decision mainly includes judging whether the current task needs to be offloaded and selecting the offloading method and the offload strategy according to the situation. Resource allocation mainly involves the reasonable allocation of limited computing, storage, bandwidth, and other resources according to task requirements and the realization of the offloading goal. Offloading goals usually include the goal of shortening the delay, the goal of reducing energy consumption, and the goal of optimizing the weighted sum of delay and energy consumption. In this paper, in order to solve the computational offloading problem in IIoT, we propose a Double-Dueling DQN (D3QN) algorithm, which is based on Deep Q-Networks (DQN) and combines the Double DQN algorithm and the Dueling DQN algorithm, aiming to optimize the weighted sum of latency and energy consumption in IIoT.

The main contributions are as follows.

1) We construct a ‘‘cloud-edge-device’’ collaborative task offload system in IIoT and define the system's communication and offload models. We separately discuss and model different processing methods, such as local computing, edge computing, and cloud computing. Finally, a mixed integer linear programming problem is developed. For this non-deterministic polynomial hard (NP-hard) problem, we use a Reinforcement Learning (RL) approach to solve it.
2) We analyze the disadvantages of the inaccuracy of the Double DQN algorithm and the overfitting of the Dueling DQN algorithm. In order to overcome their disadvantages, we propose to combine the network update method of the Double DQN algorithm with the dual-branch network structure of the Dueling DQN algorithm based on the structure of the DQN algorithm to generate the D3QN algorithm.
3) We design simulation experiments to prove that the D3QN algorithm is significantly better than the basic algorithm in reducing the total system cost. And we test the algorithm under different system model parameters to evaluate the algorithm's performance.

The rest of this paper is organized as follows. In Section II, we present related research. The system model of this paper is established, and the optimization problem is proposed in Section III. Next, the shortcomings of the original algorithm are illustrated, and the D3QN algorithm is proposed in Section IV. In Section V, we present experimental and performance evaluation results. Finally, conclusions are drawn in Section VI.

II. RELATED WORKS

In recent years, more and more scholars have conducted research on computing offloading and resource optimization in MEC. Benefiting from the development of online optimization algorithms, Meng *et al.* [6] proposed a task offloading algorithm based on particle swarm optimization to maximize the revenue of all MEC servers in the network. In [7], the authors combined the adaptive genetic algorithm with the adaptive particle swarm optimization algorithm and proposed an improved hierarchical adaptive search algorithm, which refined the resource allocation scheme layer by layer so as to minimize energy consumption. The authors in [8] proposed a delay-aware offloading scheme for cooperative Non-Orthogonal Multiple Access (NOMA)-based near-and-far MEC networks. Minimizing latency by optimally allocating transmit power, time slots, and computational tasks. A multi-objective joint optimization method for communication-computing-caching resources was proposed in [9] and was solved by the multi optimization method of nondominated sorting genetic algorithm II (NSGA-II). The above studies mainly apply traditional intelligent algorithms to computational offloading and resource optimization problems, but this class of methods often requires access to environmental a priori information, which is somewhat difficult in MEC systems.

Some scholars have combined game theory with MEC [10], [11], [12]. Huang *et al.* [10] formulated a nonlinear non-convex delay optimization offloading problem based on non-cooperative game theory and proposed a distributed task offloading algorithm to solve it. Zhao *et al.* [11] divided the computing offloading game into intra-alliance scheduling and inter-alliance formation and proposed a greedy algorithm to help users choose different computing service providers to obtain better computing resources and reduce latency. Yang *et al.* [12] formulated the proposed cost minimization problem as a strategy game. Then a potential game offloading algorithm was proposed for this problem to reach Nash equilibrium.

There are also some researchers who consider solving the target problem by applying mathematical optimization methods. The authors in [13] considered an energy-efficient NOMA-based MEC design for a multi-cell network and formulated the problem into a mathematical problem of mixed integer nonlinear programming. The problem was further decomposed into multiple sub-problems and solved in turn through mathematical optimization. In [14], the authors constructed an offload revenue function for task offloading and resourced allocation to maximize the system revenue and introduced Lyapunov optimization theory and convex optimization methods to solve the optimal solution. Hu *et al.* [15] considered the trade-off between energy efficiency and service delay, using convex decomposition and submodular methods for the proposed optimization problem and proposed an online offloading and resource allocation algorithm to achieve optimization of energy efficiency and service latency. Rafiq *et al.* [16] proposed a nonlinear integer optimization problem with the goal of minimizing task execution latency and energy consumption, decomposed the problem into interrelated sub-problems, and solved the sub-problems to obtain the optimal solution. Tang *et al.* [17] decomposed the optimization problem into two subproblems, which were solved alternately in each iteration using different algorithms until a convergence condition was reached. It can be seen that mathematical optimization is a promising optimization method, but the Lyapunov optimization method often has certain limitations and can only obtain approximate optimal solutions.

To address these challenges, more and more researchers have begun to try to solve optimization problems using deep learning methods. In [18], the authors combined Q-learning with deep neural networks to propose a sleep scheduling scheme based on Deep Reinforcement Learning (DRL) to reduce system energy consumption by automatically sleeping idle devices. Zhang *et al.* [19] divided the action space into two parts and presented a hybrid decision-based DRL algorithm, where Deep Deterministic Policy Gradients (DDPG) and DQN processing were used for continuous and discrete spaces, respectively. The authors in [20] proposed an offloading algorithm based on a meta-Reinforcement Learning (meta-RL) model and a DRL model, which improved the offloading performance in terms of latency and energy consumption during the offloading process. Yang *et al.* [21] presented a DQN-based RL algorithm for computing resource allocation with the goal of minimizing energy consumption. In [22], the authors described the computational resource allocation problem with a Markov Decision Process (MDP) model with discrete-continuous mixed action spaces and then proposed a Parametric Deep Q-Network (PDQN)-based approach to solve it. Both [23] and [24] adopted a DRL-based allocation scheme to obtain a real-time efficient allocation strategy. The authors in [25] modeled the server selection decision as the MDP with the goal of minimizing the overall cost and proposed a DRL-based algorithm to address this problem. Li *et al.* [26] presented a computational offloading model for heterogeneous network scenarios and designed a DDPG-based algorithm to minimize task latency. In [27], the authors improved on RL and proposed the RL-based state-action-reward-state-action (RL-SARSA) algorithm to make offloading decisions to optimize latency and energy consumption. The above studies are all based on the DQN architecture, which can make offloading decisions without obtaining prior information. Although the above work has made certain improvements to the deep Q-learning algorithm and greatly improved its efficiency, it has not overcome the inherent problems of over-fitting and over-estimation of Q-learning itself. In order to solve this problem, we build a "cloud-edge-device" collaborative task offloading model in IIoT application scenarios and propose a D3QN algorithm that combines Double DQN and Dueling DQN with the goal of minimizing system cost and limited computational and storage resources. The experimental simulation results show that the D3QN algorithm can optimize the objective function with limited computational resources, and the effect is significantly better than the DQN algorithm, Double DQN algorithm, and Dueling DQN algorithm.

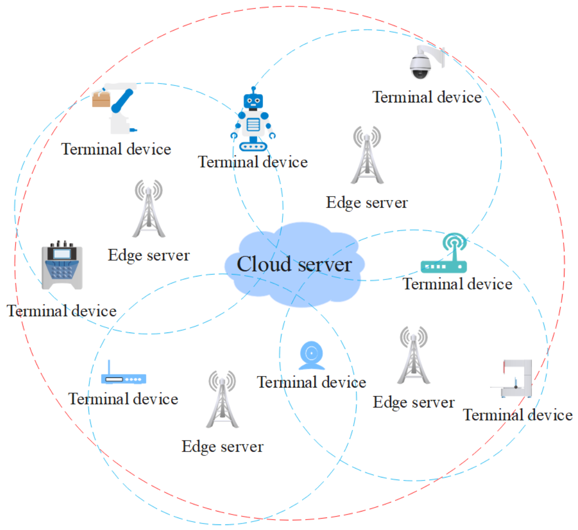

FIGURE 1. **A smart factory model.**

III. SYSTEM MODEL

A. SCENARIO DESCRIPTION

As shown in Fig. 1, in this section, we build a smart factory application scenario and propose the system architecture of edge computing, which is mainly composed of three parts:

1) Cloud server. It has powerful computing and caching capabilities, but it is often far away from the terminal device. Multiple edge servers are deployed within the coverage of cloud servers for edge computing.

2) Edge servers. The caching and computing capabilities of edge servers are much lower than cloud servers. A cloud server is connected to the edge servers through the wired optical fiber. Each edge server can only provide edge computing service to the terminal devices within its coverage area.

3) Terminal devices. It mainly includes robotic arms, instruments, machine tools, robots, cameras, etc., in the smart factory. The terminal device itself has very limited computing power. Terminal devices can process tasks by themselves or choose to offload tasks to edge servers or cloud servers for processing.

There are N terminal devices and M edge servers in our system, their sets are denoted as $\mathcal{N} = \{1, 2, \cdots, n, \cdots, N\}$, $\mathcal{M} = \{1, 2, \cdots, m, \cdots, M\}$, respectively. Same as the existing work [28], in this paper, we consider tasks to be inseparable due to the correlation between tasks, i.e., the offloading method is coarse-grained offloading.

The maximum queue length for each edge server to perform tasks is L. Each terminal device has only one task. The task of the terminal device n is denoted as Q_n. Each computing task Q_n can be represented by a triple: $Q_n = \{D_n, G_n, \tau_n\}$, where D_n denotes the amount of data consumed by task Q_n, G_n denotes the number of CPU cycles consumed by computing task Q_n, τ_n denotes the maximum tolerated delay of processing task Q_n.

B. COMMUNICATIONAL MODEL

In this paper, the offloading from the terminal device to the edge server can be divided into three steps: the task transmission stage, the task processing stage, and the processing result return stage. Since the amount of data will be greatly reduced after the task is processed and generally speaking, the downlink transmission rate of the device is much faster than the uplink transmission rate of the device. Therefore, the energy consumption and delay of the data downlink phase will be much lower than that of the data uplink phase, so the energy consumption and delay of the processing result return phase are ignored in this paper.

In the task transmission stage, we use the Orthogonal Frequency Division Multiple Access (OFDMA) technologies to divide the bandwidth of the mobile edge server m into K_m channels equally. The bandwidth of each channel is randomly allocated according to the task size, and a one-to-one relationship between users and subbands is defined so as to ensure the orthogonality of uplink transmissions of users associated with the same base station.

The terminal device n transmits the task to the edge server m through the wireless channel. Suppose the channel gain between the device n and the edge server m is H_{mn}. Considering that the moving distance of the device is almost zero in the extremely short time delay of task transmission, according to Shannon's theorem, the transmission rate of the channel can be obtained as:

$$r_{mn} = B_n log_2(1 + \frac{P_n H_{mn}}{\sigma^2}), \tag{1}$$

where B_n denotes the channel bandwidth allocated to task Q_n, P_n denotes the transmission power of the task. Particularly, if the task is processed locally, $P_n = 0$. σ^2 is the noise power.

C. OFFLOADING MODEL

During the task processing of the terminal device, it can choose whether the task is executed locally or offloaded to an edge server or a cloud server for processing. We define the offloading decision for terminal device n as x_{mn}, $x_{mn} \in \{0, 1\}$, where $x_{mn} = 0$ denotes the task is not offloaded and processed locally, $x_{mn} = 1$ denote the task is offloaded to the edge server m for processing. We denote the offloading decisions of all terminal devices about edge servers as matrix $\mathcal{X}$:

$$\mathcal{X} = \begin{pmatrix} x_{11} & \cdots & x_{1n} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_{m1} & \cdots & x_{mn} & \cdots & x_{mN} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{M1} & \cdots & x_{Mn} & \cdots & x_{MN} \end{pmatrix}_{M \times N} . \tag{2}$$

When the task is neither processed locally nor on the edge server, the task is offloaded to the cloud server for processing. We define y_n as the offloading decision of the terminal device n to the cloud server, $y_n \in \{0, 1\}$, where $y_n = 0$ denotes the task is not offloaded to the cloud server for processing, otherwise $y_n = 1$. Since we define the offloading method in this paper as coarse-grained offloading, only one offloading method can be selected for the task of each terminal device n, then the constraints of the offloading decision can be expressed as:

$$\sum_{m=1}^{M} x_{mn} + y_n \in \{0, 1\}. \tag{3}$$

D. COMPUTATIONAL MODEL

1) LOCAL COMPUTING

Local computing means that the tasks of the terminal device are not offloaded and are processed locally on the terminal device. The local computing power of the terminal device is f_{local}, which can be expressed by the CPU cycle frequency of the device. It is defined that the local computing power of terminal device n remains unchanged during a scheduling period, particularly, $f_{local} > 0$. The local processing delay of task Q_n is defined as:

$$T_{local} = \frac{G_n}{f_{local}}. \tag{4}$$

Based on [29], the energy consumption of the device is proportional to the square of the frequency, so the local computing energy consumption of the terminal device can be expressed as:

$$E_{local} = \kappa_{local} G_n f_{local}^2. \tag{5}$$

In this paper, we define the total cost of the terminal device to process the task Q_n locally as the weighted sum of the required delay and energy consumption, which can be expressed as:

$$Z_{local} = \lambda_{local} T_{local} + (1 - \lambda_{local}) E_{local}, \tag{6}$$

where λ_{local} is the weight parameter of the local computing delay, $\lambda_{local} \in [0, 1]$.

2) EDGE COMPUTING

In the edge computing model, the tasks of the terminal device are offloaded to the edge server for execution. Same as the local computing model, the computing capability assigned by the edge server m to process the task of the terminal device n is f_{mn}, f_m^{max} is the maximum computing power,

$\sum_{n=1}^{N} f_{mn} \leq f_m^{max}$. C_m^{max} is the maximum caching capacity of the edge server m. Then the maximum cache matrix $\mathcal{C}$ of all edge servers is:

$$\mathcal{C} = \{C_1^{max}, C_2^{max}, \ldots, C_m^{max}, \ldots, C_M^{max}\}_{1\times M}. \tag{7}$$

The delay and energy consumption of the task transmitted from the terminal device n to the edge server m are expressed as:

$$T_{mn} = \frac{D_n}{r_{mn}}, \tag{8}$$
$$E_{mn} = P_{mn}T_{mn}, \tag{9}$$

where P_{mn} is the uplink transmission power of the terminal device n transmitting the task to the edge server m. The delay and energy consumption required by the edge server to process the task is expressed as:

$$T_{exe} = \frac{G_n}{f_{mn}}, \tag{10}$$
$$E_{exe} = \mu G_n f_{mn}^2. \tag{11}$$

Therefore, the total delay and energy consumption required by the edge server m to process the task of the terminal device n can be expressed as:

$$T_{mec} = T_{mn} + T_{exe}, \tag{12}$$
$$E_{mec} = E_{mn} + E_{exe}. \tag{13}$$

Similarly, the total cost of offloading tasks to edge servers is defined as the weighted sum of its delay and energy consumption, which can be expressed as:

$$Z_{mec} = \lambda_{mec}T_{mec} + (1 - \lambda_{mec})E_{mec}, \tag{14}$$

where λ_{mec} is the weight parameter of delay in edge computing, $\lambda_{mec} \in [0, 1]$. During the offloading selection process of the edge server, its remaining computing power should be described. In this paper, the remaining computing power of the edge server m is expressed as:

$$f_m^{rem} = f_m^{max} - \sum_{n=1}^{N} x_{mn} f_{mn}. \tag{15}$$

The remaining computing power of all edge servers can be represented by matrix $\mathcal{F}$:

$$\mathcal{F} = \{f_1^{rem}, f_2^{rem}, \ldots, f_m^{rem}, \ldots, f_M^{rem}\}_{1\times M}. \tag{16}$$

3) CLOUD COMPUTING

If the remaining computing resources of the edge server are less than those required by the current task, the system will offload the current task to the cloud server for processing. Different from the edge computing model, cloud computing has negligible delay and energy consumption due to its powerful computing power. Therefore, the delay and energy consumption required in the cloud computing model is mainly concentrated in the uplink transmission phase of the task. r_{cn} and P_{cn} are the transmission rate and transmission power of the terminal device n task to the cloud server, respectively. The time delay T_{cc} and the energy consumption E_{cc} in the process can be expressed separately as:

$$T_{cc} = \frac{D_n}{r_{cn}}, \tag{17}$$
$$E_{cc} = P_{cn}T_{cc}. \tag{18}$$

Similarly, in the cloud computing model, the total cost of the system can be expressed as:

$$Z_{cc} = \lambda_{cc}T_{cc} + (1 - \lambda_{cc})E_{cc}, \tag{19}$$

where λ_{cc} is the weight parameter of delay in cloud computing, $\lambda_{cc} \in [0, 1]$.

E. PROBLEM DESCRIPTION

We aim to minimize the weighted sum of delay and energy consumption. According to the above three computing models, it can be obtained that for terminal device n, the total cost of processing tasks in this system is expressed as:

$$Z_n^{sum} = (1 - y_n)((1 - \sum_{m=1}^{M} x_{mn})Z_{local} + \sum_{m=1}^{M} x_{mn}Z_{mec}) + y_n Z_{cc}. \tag{20}$$

Based on the above discussion, we propose an optimization problem, which aims to minimize the delay and energy consumption of the system processing tasks, and the optimization problem is expressed as:

$$\begin{aligned} P:\ & \min_{\mathcal{X},\mathcal{C},\mathcal{F}} \sum_{n=1}^{N} Z_n^{sum} \\ s.t.\ & C1: x_{mn} \in \{0, 1\}, \quad \forall m \in M, n \in N \\ & C2: y_n \in \{0, 1\}, \quad \forall n \in N \\ & C3: \sum_{m=1}^{M} x_{mn} + y_n \in \{0, 1\} \\ & C4: f_m^{rem} \geq 0 \\ & C5: \sum_{n=1}^{N} x_{mn} \leq K_m, \quad \forall m \in M \end{aligned} \tag{21}$$

where $\mathcal{X}, \mathcal{C}$ and $\mathcal{F}$ are the offloading decision set, the edge server caching capability set, and the edge server computing power set, respectively. $C1$ and $C2$ respectively represent the edge offloading decision and cloud offloading decision as two-dimensional variables. $C3$ limits the task of each terminal device to only one processing method. $C4$ indicates that the current remaining resources of the edge server m are non-negative. $C5$ defines that the number of tasks assigned to each edge server m cannot exceed the number of its bandwidth.

IV. ALGORITHM DESIGN

For the optimization problem (21) obtained in Section III, the problem is a mixed-integer nonlinear programming problem, which is NP-hard. Solving the objective function can be achieved by finding the optimal offloading decision set $\mathcal{X}$ under the conditions of the edge server caching capability set $\mathcal{C}$ and the edge server computing power set $\mathcal{F}$. Since the process of solving the optimization problem requires not only satisfying the optimal policy problem for task offloading but also making each edge server fully utilize its limited resources, we believe that this problem can be solved by RL algorithms. The Double DQN and Dueling DQN algorithms based on the DQN algorithm architecture improve the training algorithm as well as the model structure of DQN, respectively, but the single algorithm still has certain disadvantages. Therefore, we combine the Double DQN and Dueling DQN algorithms to propose an improved D3QN algorithm. The D3QN algorithm changes the method of updating action values to avoid problems such as overfitting. Meanwhile, the D3QN algorithm is able to partition the original single structure of DQN and calculate the outcomes jointly with the value function and the advantage function, which improves the accuracy of the results. We will discuss the disadvantages of the Double DQN algorithm and the Dueling DQN algorithm, respectively, in this section and show how the D3QN algorithm overcomes these disadvantages.

A. DOUBLE DQN ALGORITHM

The Double DQN algorithm structure is similar to DQN, but it contains two Q functions. Each Q function can use the value of the other Q function to update the next state. The update method is as follows: Double DQN first finds the action corresponding to the maximum Q value in the current Q network and then places the selected action with the maximum value in the target network to calculate the target Q value. The Q-value is updated as follows:

$$y = r + \gamma Q_2(\phi(s'), \arg\max_{a'} Q_1(\phi(s'), a, w_1), w_2), \quad (22)$$

where r is the immediate reward, Q_1 is the current network, and Q_2 is the target network. w_1 and w_2 are the current network parameters and the target network parameters, respectively. γ is the attenuation factor. a is defined as the action, a' is the next action, and $\phi(s')$ is the next state. The specific process of the Double DQN algorithm is shown in Fig. 2.

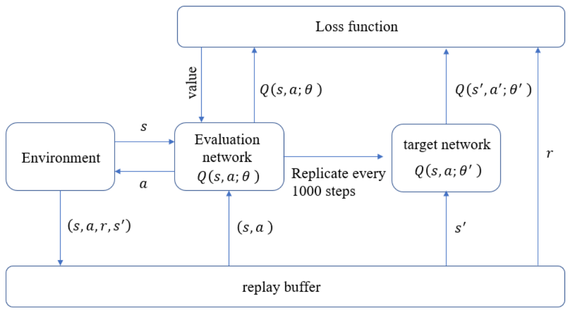

FIGURE 2. The structure of double DQN.

By analyzing the network structure of Double DQN, we can find that its output is only related to the action value of each action in this state. Such a calculation is incomplete, as mentioned in [30]. In some cases, the state value has a huge impact on the Q value, but the action advantage value has no effect on the Q value. At this time, the calculation method of Double DQN is accurate. But in another case, both the state value and the action advantage have a large impact on the Q value. In this case, since Double DQN inherits the single-branch network structure of the basic DQN, it cannot obtain the action value of each action through the state value and advantage value, which causes the inaccuracy of the results.

B. DUELING DQN ALGORITHM

The Dueling DQN algorithm improves the network structure of the DQN algorithm and divides the traditional output layer into two parts: the value function network and the advantage function network. The linear combination of these two parts constitutes the final output of the Q network. Then, it is updated as follows:

$$Q(s, a, w, \alpha, \beta) = V(s, w, \alpha) + (A(s, a, w, \beta) - \frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A(s, a', w, \beta)), \quad (23)$$

where s is defined as the state, and w denotes the parameters of the convolutional layers. α and β are the parameters of the two streams of fully-connected layers. $|\mathcal{A}|$ denotes the number of selectable actions. $V(s, w, \alpha)$ is the state-value function, $A(s, a, w, \beta)$ is the action-advantage function.

Since Dueling DQN inherits the method of updating action values from the basic DQN algorithm, it still has the same problem of maximizing deviation as the DQN algorithm [31], which tends to make the estimated action value too large.

C. HYBRID D3QN ALGORITHM

Considering the shortcomings of the above two algorithms, we combine the Double DQN and the Dueling DQN to form the D3QN algorithm. The D3QN algorithm inherits the double network structure of the Double DQN algorithm, which includes an evaluation network Q_1 and a target network Q_2. The evaluation network Q_1 is used to select the next action $a^{next} = \arg\max_{a'} Q_1(\phi(s'), a, w_1)$. According to Equation (22), we evaluate a^{next} through the target network Q_2 to reduce the over-estimation problem. Then, the D3QN algorithm combines the network structure of the Dueling DQN algorithm, adding the state-value function $V(s)$ and the action-advantage function $A(s, a)$ between the last hidden layer and the output layer, where $V(s)$ is only related to the state s, and $A(s, a)$ is affected by the state and action. The optimal action-value function $Q(s, a)$ is used to represent the linear combination of these two parts, and its specific calculation method is shown in Equation (23). This enables the D3QN algorithm to get the value of each action by the value of the state and the value of advantage, resulting in more accurate results. The network structure of the D3QN algorithm is shown in Fig. 3, where $s = \{s_1, s_2, \ldots, s_n, \ldots, s_N\}$ is the state space of the entire system, which shows the system state of each terminal device n, $s_n = \{f_{local}, r_{mn}, P_{mn}, D_n, G_n, \tau_n, K_m, f_m^{rem}, C_m^{rem}, r_{cn}, P_{cn}\}$.

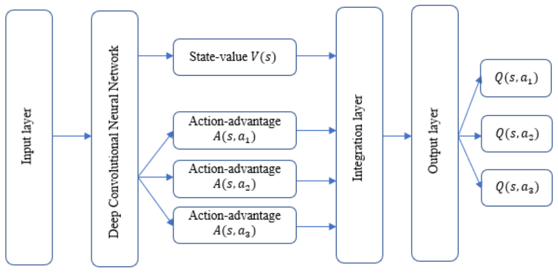
Input layer
Deep Convolutional Neural Network
State-value $V(s)$
Action-advantage $A(s,a_1)$
Action-advantage $A(s,a_2)$
Action-advantage $A(s,a_3)$
Integration layer
Output layer
$Q(s,a_1)$
$Q(s,a_2)$
$Q(s,a_3)$

FIGURE 3. Neural network architecture framework of D3QN.

$a = \{a_1, a_2, \ldots, a_n, \ldots, a_N\}$ is the action space of the entire system, $a_n = \{x_{mn}, y_n\}$ is the way to handle task selection. When the system performs an action a_n in the state s_n, it will receive the corresponding reward. The goal of reinforcement learning is to maximize the reward function, so we define the reward function as:

$$r = -\sum_{n=1}^{N} Z_n^{sum}. \tag{24}$$

D3QN obtains the optimal policy by performing Algorithm 1.

V. SIMULATION RESULTS

According to the smart factory application model constructed in Section III, we configure the model environment parameters according to the actual scenario requirements. The specific parameters are shown in Table 1.

TABLE 1. Simulation parameters of system environment.

Parameter	Value
Task data size	200–300Kbit
Terminal device computing power	15–20GHz/sec
Edge Server Computing Power	50–70GHz/sec
Edge Server Caching	1500MB
Bandwidth	40MHz
Distance between terminal devices and edge servers	100–300KM

With the edge server as the center, the terminal devices are randomly distributed within 300 meters to 1000 meters around it. The channel gain is denoted as $H_k = 127 + 30 \times \log_2 d_n$, where d_n is the distance from the terminal device to the edge server.

The D3QN algorithm proposed in this paper is based on improving the Double DQN algorithm and the Dueling DQN algorithm. Therefore, in the experimental comparison, these two algorithms are introduced to compare with the D3QN algorithm. Then, to calculate the D3QN algorithm compared with the two improved DQN algorithms for the optimization degree of the basic DQN algorithm, this paper introduces the DQN algorithm to conduct comparative experiments simultaneously. The experimental result is shown in Fig. 4.

Algorithm 1 D3QN
```
1: Initialize the system model environment, Initialize replay buffer $\mathcal{D}$
2: Initialize the evaluation network $Q_1$ with random parameters $w_1$
3: Initialize the target network $Q_2$ with random parameters $w_2$, $w_2 = w_1$
4: Fill the replay buffer $\mathcal{D}$ with the data generated by the random policy until it is full
5: while True do
6:     Reset system model environment
7:     for terminal devices $n = 1$ to $N$ do
8:         Training evaluation network $Q_1$, Update the target network $Q_2$ with the next state condition
9:         Randomly sample Mini_Batch form $\mathcal{D}$
10:        Select the next action $a^{next} = \arg\max_{a'} Q_1(\phi(s'), a, w_1)$
11:        Set $y = r + \gamma Q_2(\phi(s'), a^{next}, w_2)$
12:        Use the loss function to train $Q_1$ to update the parameter $w_1$
13:        Update the parameter, $w_2 = w_1$
14:    end for
15:    if $Q_1$ is converged then
16:        $Q_2 = Q_1$
17:        $Q_{opt} = Q_2$
18:    end if
19: end while
```

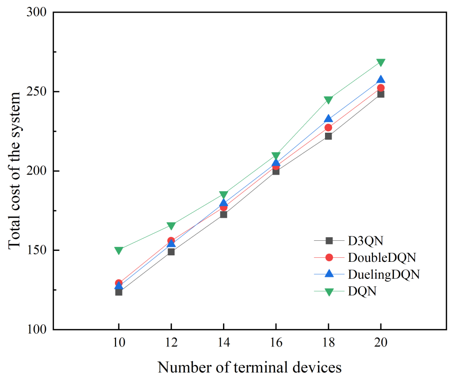

FIGURE 4. The relationship between the total system cost and the number of terminal devices under different algorithms.

In the above experiments, we fixed the number of edge servers to 10 and compared the total system cost of different algorithms for different numbers of terminal devices. We record the results in Table 2. As can be seen from Figure 4, the D3QN-based offloading strategy proposed in this paper is superior to the other three DQN algorithms. The data in Table 2 enables us to see the improvement of the D3QN algorithm over the original algorithm more intuitively. Taking the number of terminal devices equal to 10 as an example, we set the total system cost of the DQN algorithm to be 1, then the total system cost of the Double DQN algorithm is 0.859.

TABLE 2. Total system cost under different algorithms (Normalizing the total system cost of the DQN algorithm).

Number of terminal devices	DQN	Double DQN	Dueling DQN	D3QN
num = 10	1	0.859	0.847	0.822
num = 12	1	0.941	0.927	0.898
num = 14	1	0.955	0.968	0.930
num = 16	1	0.966	0.976	0.951
num = 18	1	0.927	0.948	0.905
num = 20	1	0.938	0.957	0.924

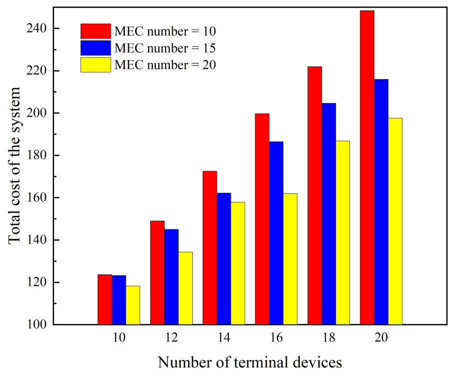

FIGURE 5. The relationship between the total system cost and the number of terminal devices under different numbers of edge servers.

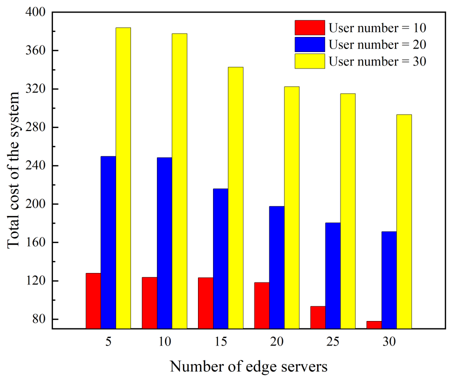

FIGURE 6. The relationship between the total system cost and the number of edge servers under different numbers of terminal devices.

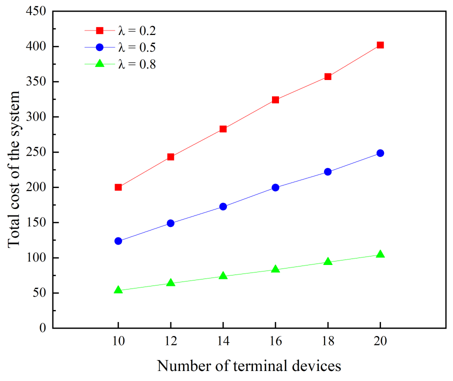

FIGURE 7. The relationship between the total system cost and the number of terminal devices under different weight parameters.

Through the calculation, the improvement rate of the Double DQN algorithm to the DQN algorithm can be obtained by $((1-0.859)/1)\times 100\% = 14.1\%$. Similarly, the improvement rate of the Double DQN algorithm to the DQN algorithm can be obtained by $((1 - 0.847)/1) \times 100\% = 15.3\%$. At this time, the improvement rate of the D3QN algorithm to the DQN algorithm can be obtained by $((1 - 0.822)/1) \times 100\% = 17.8\%$. Our algorithm performs better in terms of total system cost, regardless of changing environmental conditions. Therefore, it can be concluded that the optimization effect of the D3QN algorithm proposed in this paper is obviously better than that of the other three DQN algorithms, avoiding problems such as over-fitting, overestimation, and inaccuracy.

As shown in Fig. 5, we compared the relationship between the total system cost and the number of terminal devices under the condition of different numbers of edge servers. It can be seen that, with the increase in the number of edge servers, the overall system cost is decreasing trend. This is because more edge servers provide the system with more computing resources, and the terminal device has a greater possibility of selecting a more suitable edge server for offloading tasks. When each terminal device has a better choice, the total cost of the system will be reduced.

As shown in Fig. 6, we compared the impact of different numbers of terminal devices on the total system cost. By changing the number of edge servers, it can be seen that when the number of edge servers increases, the total cost of the system will decrease. And in the scenario of the same number of edge servers, as the number of terminal devices increases, the total system cost will also increase. This is because the more terminals there are, the more computing resources are required. It can be seen from Figure 6 that if the number of edge servers is more than the number of terminal devices, the system cost will be greatly reduced. This is because more edge servers provide the system with sufficient computing resources to ensure that each terminal device selects the optimal offloading policy.

In order to compare the impact of delay and energy consumption on the total system cost, we conducted a comparative experiment by modifying the weight parameter, and the experimental results are shown in Fig. 7. λ is the weight parameter of the delay cost in the total system cost. From the vertical comparison, it can be seen that the smaller the proportion of delay cost, the greater the total system cost. Otherwise, the total cost of the system is smaller. This shows that in the same scenario, the system processing task delay cost is lower than the energy cost. Therefore, selecting a suitable edge server for task offloading and reducing the task computing cost of the system is the key to reducing the total system cost.

VI. CONCLUSION

In this paper, we studied the task offloading and resource allocation strategies for MEC in the current IIoT environments. First, we proposed an edge computing model based on the smart factory framework, and an NP-hard problem is proposed with the goal of minimizing the total system cost. Then according to the characteristics of the objective function, we proposed a D3QN algorithm to solve the over-fitting and over-estimation problems of the DQN algorithm itself. Finally, the results of simulation experiments demonstrate that our algorithm was better than the DQN, the Double DQN, and the Dueling DQN algorithms in reducing the total system cost. In future work, we will apply the D3QN algorithm in environments that are more sensitive to system cost and reduce the total system cost by further reducing the energy consumption in the computing process.

REFERENCES

[1] A. Rojko, "Industry 4.0 concept: Background and overview," *Int. J. Interact. Mobile Technol.*, vol. 11, no. 5, pp. 77–90, Nov. 2017.

[2] J. Weng, F. Wei, A. Jaiswal, and B. Noche, "A review of industry 4.0 on national level and a concept of industry 4.0 stages based on technical level," in *Proc. 17th Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Coral Bay, Pafos, Cyprus, Jul. 2021, pp. 252–258.

[3] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, "Deep learning for edge computing applications: A state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58322–58336, 2020.

[4] W. Chen, X. Qiu, T. Cai, H.-N. Dai, Z. Zheng, and Y. Zhang, "Deep reinforcement learning for Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1659–1692, 2021, doi: 10.1109/COMST.2021.3073036.

[5] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, 4th Quart., 2021, doi: 10.1109/COMST.2021.3106401.

[6] Y. Meng and J. Li, "Task offloading and resource allocation mechanism of moving edge computing in mining environment," *IEEE Access*, vol. 9, pp. 155534–155542, 2021, doi: 10.1109/ACCESS.2021.3129464.

[7] T. Zhou, Y. Yue, D. Qin, X. Nie, X. Li, and C. Li, "Joint device association, resource allocation and computation offloading in ultra-dense multi-device and multi-task IoT networks," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 18695–18709, Oct. 2022, doi: 10.1109/JIOT.2022.3161670.

[8] T. He, D. Wang, F. Zhou, W. Liang, X. Tang, and D. Zhai, "Delay-aware offloading for cooperative NOMA-based near-and-far MEC networks," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Xiamen, China, Jul. 2021, pp. 978–983.

[9] X. Wang, W. Cheng, and C. Ren, "Multi-objective joint optimization of communication-computation-caching resources in mobile edge computing," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Xiamen, China, Jul. 2021, pp. 2762–2773.

[10] J. Huang, M. Wang, Y. Wu, Y. Chen, and X. Shen, "Distributed offloading in overlapping areas of mobile edge computing for Internet of Things," *IEEE Internet Things J.*, vol. 26, no. 6, pp. 2762–2773, Jan. 2018, doi: 10.1109/JIOT.2022.3143539.

[11] J. Zhao, "Offloading selection based on heterogeneous utility in MEC networks: A coalition formation game-theoretic approach," in *Proc. IEEE 6th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Chengdu, China, Apr. 2021, pp. 469–472.

[12] L. Yang, H. Zhang, X. Li, H. Ji, and V. C. M. Leung, "A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2762–2773, Dec. 2018, doi: 10.1109/TNET.2018.2876941.

[13] B. Liu, C. Liu, and M. Peng, "Resource allocation for energy-efficient MEC in NOMA-enabled massive IoT networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 4, pp. 1015–1027, Apr. 2021, doi: 10.1109/JSAC.2020.3018809.

[14] S. Xia, X. Wen, Z. Yao, Y. Li, and G. Wang, "Dynamic task offloading and resource allocation for heterogeneous MEC-enable IoT," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Chongqing, Ching, Aug. 2020, pp. 847–852.

[15] H. Hu, W. Song, Q. Wang, R. Q. Hu, and H. Zhu, "Energy efficiency and delay tradeoff in an MEC-enabled mobile IoT network," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15942–15956, Feb. 2022, doi: 10.1109/JIOT.2022.3153847.

[16] A. Rafiq, W. Ping, W. Min, S. H. Hong, and N. N. Josbert, "Optimizing energy consumption and latency based on computation offloading and cell association in MEC enabled industrial IoT environment," in *Proc. 6th Int. Conf. Intell. Comput. Signal Process. (ICSP)*, Xi'an, Ching, Apr. 2021, pp. 10–14.

[17] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Task offloading and caching for mobile edge computing," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Harbin, China, Jun. 2021, pp. 698–702.

[18] N. Zhu, X. Xu, S. Han, and S. Lv, "Sleep-scheduling and joint computation-communication resource allocation in MEC networks for 5G IIoT," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Nanjing, China, Mar. 2021, pp. 1–7.

[19] J. Zhang, J. Du, Y. Shen, and J. Wang, "Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9303–9317, Oct. 2020, doi: 10.1109/JIOT.2020.3000527.

[20] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "MR-DRO: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments," *IEEE Internet Things J.*, early access, Nov. 8, 2021, doi: 10.1109/JIOT.2021.3126101.

[21] Y. Yang, Y. Hu, and M. C. Gursoy, "Deep reinforcement learning and optimization based green mobile edge computing," in *Proc. IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2021, pp. 1–2.

[22] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, and Y. Yang, "Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing," *IEEE Trans. Mobile Comput.*, early access, Feb. 4, 2022, doi: 10.1109/TMC.2022.3148254.

[23] Z. Wu and D. Yan, "Deep reinforcement learning-based computation offloading for 5G vehicle-aware multi-access edge computing network," *China Commun.*, vol. 18, no. 11, pp. 26–41, Nov. 2021, doi: 10.23919/JCC.2021.11.003.

[24] X. Li, H. Yang, Q. Yao, B. Bao, J. Li, and J. Zhang, "Deep reinforcement learning-based power and caching joint optimal allocation over mobile edge computing," in *Proc. IEEE Int. Sympo Broadband Multimedia Syst. Broadcast. (BMSB)*, Chengdu, China, Oct. 2020, pp. 1–3.

[25] H. Liu and G. Cao, "Deep reinforcement learning-based server selection for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13351–13363, Dec. 2021, doi: 10.1109/TVT.2021.3124127.

[26] Y. Li, F. Qi, Z. Wang, X. Yu, and S. Shao, "Distributed edge computing offloading algorithm based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 85204–85215, 2020, doi: 10.1109/ACCESS.2020.2991773.

[27] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020, doi: 10.1109/ACCESS.2020.2981434.

[28] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022, doi: 10.1109/TMC.2020.3036871.

[29] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. 2nd USENIX Workshop Hot Topics Cloud Comput.*, Boston, MA, USA, 2010, pp. 1–7.

[30] Z. Wang and N. D. M. Freitas Lanctot, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1995–2003.

[31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, Phoenix, AR, USA, 2016, pp. 2094–2100.

WEIJUN CHENG (Member, IEEE) received the M.S. degree in electronics and control engineering from the China University of Mining and Technology, Beijing, China, in 1998, and the Ph.D. degree in telecommunications engineering from the Beijing University of Posts and Telecommunications, Beijing, in 2004. From 2005 to 2007, he was a Postdoctoral Research Fellow in electronics engineering with Peking University, Beijing. From October 2017 to October 2018, he was a Visiting Scholar at the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA, along with Prof. Junshan Zhang. He is currently an Associate Professor at the School of Information Engineering, Minzu University of China, Beijing. His research interests include wireless communication theory and AI in the IoT.

XIAOSHI LIU received the B.S. degree in mathematics and applied mathematics from the Shandong University of Science and Technology, Qingdao, China, in 2021. He is currently pursuing the M.S. degree in electronic information with the Minzu University of China, Beijing, China. His research interests include mobile edge computing, deep reinforcement learning, and convex optimization.

XIAOTING WANG received the B.S. degree in software engineering from Qingdao University, Qingdao, China, in 2019. She is currently pursuing the M.S. degree in computer science and technology with the Minzu University of China, Beijing, China. Her research interest includes mobile edge computing.

GAOFENG NIE (Member, IEEE) received the B.S. degree in communications engineering and the Ph.D. degree in telecommunications and information system from the Beijing University of Posts and Telecommunications (BUPT), in 2010 and 2016, respectively. He is currently a Lecturer with the BUPT. His research interests include SDN over wireless networks and key technologies in 5G/B5G wireless networks.