Journal of Combinatorial Mathematics
and Combinatorial Computing
www.combinatorialpress.com/jcmcc

Combinatorial Press

An optimal scheduling method for industrial internet resources based on distributed computing and cloud computing platforms

Xingyan Shi[1,] ✉

[1] *Faculty of Information Engineering, Henan Vocational College of Agriculture, Zhengzhou, Henan, 451450, China*

ABSTRACT

Industrial Internet based on distributed computing and cloud computing platform forms a "cloud-edge-end" cooperative system. Facing the problem of computing task offloading for machine-type communication devices in industrial Internet scenarios, this paper transforms the task offloading problem into a Markov decision process problem, proposes an online task offloading algorithm based on deep Q neural network (DQN), and designs an optimal scheduling method based on iterative optimization for industrial Internet resources. Simulation experiments are conducted by comprehensively considering the network environment and server state during the task offloading process, and compared with other resource optimization scheduling strategies. The results show that the DQN algorithm converges in about 9000 steps and has good convergence performance. The offloading strategy based on the DQN algorithm can effectively reduce the delay, energy consumption and total overhead of the computational task offloading system in the economy.

Keywords: Industrial Internet, Distributed Computing, Cloud Computing, Markov Decision Making, DQN

1. Introduction

The development of social economy and information technology has introduced Internet technology to various industries, and the industrial field is also relying on Internet technology to make more and more progress. For enterprises with production and operation characteristics such as a large number of materials, diverse product types, and a large amount of data information, it is crucial to ensure data stability, security, and timeliness [1-3]. Distributed computing and cloud computing platform for industrial Internet resource scheduling injected a new impetus, in-depth study of its integration and configuration has become the next key direction for enterprises [4].

✉ Corresponding author.
E-mail address: 13849029903@163.com (X. Shi).
Received 10 June 2024; Revised 25 August 2024; Accepted 20 January 2025; Published Online 16 April 2025.
DOI: 10.61091/jcmcc127b-189

With the evolution and development of industrial Internet platform technology, the traditional central platform as the core of the construction program is limited by the massive access to data, bandwidth, latency and other factors, is showing a new form of industry [5-7]. The edge-cloud cooperative architecture with "edge computing platform" and "center cloud platform" as the core is becoming a typical architectural composition of industrial Internet platform [8-9]. Edge computing, as a typical distributed computing paradigm, can significantly reduce network transmission delay and effectively alleviate network congestion and other problems, as well as reduce the network load overhead, although it is a little short in computing power [10-11]. Edge computing and cloud computing belong to a complementary relationship, cloud computing can access the historical data of edge computing, the advantage of edge computing is that it can process real-time data faster, while cloud computing is more suitable for processing large amounts of data [12-15]. Distributed computing methods are gradually becoming an important technical means to enhance the service capability of cloud platforms, reduce energy consumption, and achieve reliable Internet services, due to meeting the key requirements of efficient connectivity, fast computational response, data optimization, application intelligence, and security and privacy protection in the process of industrial digitization [16-19].

In order to cope with the problems of limited computing power and network congestion of industrial Internet devices, it has become a popular research issue nowadays to process the computing tasks generated by industrial Internet nodes with the smallest possible energy consumption, economy, and other overheads without exceeding the load of edge computing servers. Sun, Z. et al. proposed a joint offloading optimization scheme for the problem of limited edge computing resources and capabilities in industrial internet scenarios, which effectively reduces the latency of industrial internet computing services and minimizes the task offloading failure rate by predicting the edge resource occupancy to formulate task offloading policies [20]. Yang, C. et al. assembled a variety of algorithms to optimize the time-sensitive data traffic control problem that arises during the complex manufacturing task allocation process in the Industrial Internet, preventing the emergence of critical network congestion between the endpoints and the cloud due to the transmission of a large number of data services [21]. Okwuibe, J. et al. proposed a resource management scheme, SDRM, that applies a software-defined approach to centrally manage all the resources in a cloud platform and dynamically adjusts the allocated resources in the industrial Internet through constraints to achieve dynamic load balancing and efficient utilization of edge cloud resources [22]. Laili, Y. et al. established a cloud-edge cooperative task scheduling model based on a combinatorial and evolutionary algorithm, which is able to quickly assign tasks to cloud servers and edge servers and reduce the huge energy consumption caused by device communication [23]. Wang, Y. et al. examined the application of end-edge cloud collaborative computing approach in industrial IoT resource task management by formulating a multi-intelligence deep deterministic policy gradient to participate in collaborative computation and resource allocation problem, which reduces the latency and energy consumption of task offloading process [24]. Ma, J. et al. showed that cloud or edge computing paradigm alone cannot solve the dynamic fluctuation of orders due to personalized customization demand, and proposed the edge-cloud cooperative production scheduling model, which is highly effective in the cloud for periodic prediction based on edge upload data and guiding the edge for real-time distributed scheduling [25]. Zhang, X. et al. pointed out that the maximum peak flow rate of industrial Internet is closely related to bandwidth reservation, and by studying the problem of minimizing the maximum peak flow rate of periodic real-time services of distributed sensors, bandwidth resources can be effectively saved and transmission delay can be reduced to avoid network congestion in network transmission [26].

In this paper, we propose a data computing task offloading system architecture for the industrial Internet with respect to the problem of data computing task offloading strategy for devices within the industrial Internet system based on distributed computing and cloud computing platform. And the computing offload scheduling model is divided into network model, delay model, energy consumption

model and economic overhead model. With the optimization objective of minimizing the total overhead of data computing tasks processed by a group of devices in a limited time, this optimization problem is modeled as an MDP, and the state space, action space, and reward function of the problem are set and defined. Iterative solution of optimal offload scheduling policy based on reinforcement learning approach. The optimization effect is verified by simulation tests on the scheduling strategy of computing resources under the architecture.

2. Industrial Internet Computing Offload Scheduling Model

Industrial Internet is one of the most important ways of digital transformation for countries around the world, which builds a comprehensive interconnection of "human-machine-object" through the use of new-generation information technology [27], and realizes real-time collection, free transmission, accurate analysis and intelligent feedback of huge amounts of industrial data. The industrial Internet based on distributed computing and cloud computing platform is to combine the centralized resource scheduling capability of cloud computing with the decentralized processing capability of edge nodes of distributed computing to form a "cloud-edge-end" collaborative system to support real-time data analysis, dynamic resource allocation and cross-domain task collaboration.

In order to cope with the problems of insufficient processing capacity of terminal equipment and limited computing resources, computation offloading technology has become one of the main response methods.

Compute offloading is a key technology initially proposed based on cloud computing architecture, which aims to provide computing resources for resource-constrained terminal devices to run compute-intensive applications, accelerate computing speed, save energy consumption, and make up for the deficiencies in resource storage, computing performance, and energy efficiency. In the early computing offloading with cloud computing as the main architecture, user devices access remote centralized cloud through the core network to offload computing tasks to the cloud. With the increase in the density and variety of computing tasks, computation offloading under the traditional cloud computing architecture leads to problems such as unpredictable latency and high transmission energy consumption. In this regard, compute offloading under the emerging MEC architecture can provide compute services to end devices more quickly and efficiently, while significantly reducing the pressure on the core network.

2.1. Distributed and Cloud Computing

Cloud computing is essentially a network that can provide computing resources, and users can obtain computing resources or storage resources from the cloud at any time according to their own needs to meet their own application requirements [28]. The main features of cloud computing are high computing power, high reliability, and centralized deployment. However, with the development of the Internet of Things and the industrial Internet, a large number of application devices access the network, the demand pattern has been transformed, and the shortcomings of cloud computing itself, which has a high network latency and high computational overhead, become more and more obvious when using cloud computing for data computation. In order to cope with today's larger network access and lower network latency and energy consumption needs, mobile edge computing has become a more popular and practical way of computing.

MEC is a distributed computing method based on mobile communication networks, which realizes more efficient and convenient computing resource utilization by deploying computing resources at the edge of the network. While the traditional computing process of cloud computing takes place in a remote central server far away from users and devices, MEC allows the computing process to take place in base stations or other nodes in the network. By transferring the load of cloud computing to various edge servers, MEC can effectively reduce network congestion and transmission delay, thus

improving user experience. Although MEC is not as good as cloud computing in terms of computational power and storage capacity, it has a great advantage in terms of transmission delay, power loss and scalability [29].

2.2. System modeling

The industrial internet network architecture model based on distributed computing and cloud computing platform proposed in this paper is shown in Fig. 1. This scenario contains N users and M WiFi nodes. At the same time, 1 base station assembled with MEC server is also deployed in the district. Assuming that the users need to process a batch of computationally intensive tasks, they can choose to process them locally or offload them to the MEC server or the central cloud server, and the maximum computational load of the MEC server is L. When the users upload the task data, they can choose to transmit it either through the mobile network or through the WiFi network.

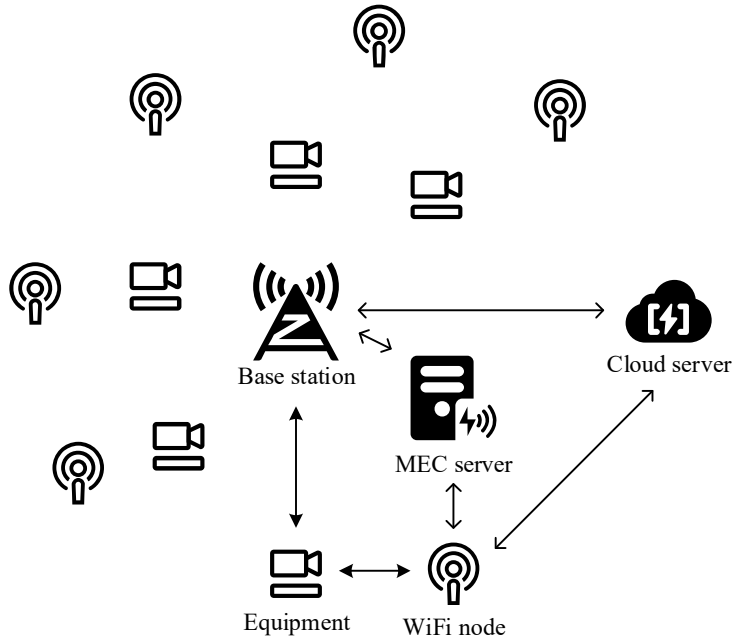


Fig. 1. System model

2.2.1. Network Layer Architecture. In a small area, it is assumed that each device can compete to get the same channel bandwidth regardless of whether a mobile or WiFi network is used. When transmitting via the mobile network, each device has a fixed rate r_b when transmitting, which is given by the following equation:

$$r_b = B_b lb(1 + \frac{p_b \cdot h_b}{B_b \cdot N_b}) \tag{1}$$

where, B_b is the channel bandwidth of the mobile network, h_b is the channel gain, and N_b is the power spectral density of the noise in the channel.

When transmitting via WiFi network, the device first searches and obtains the transmission power $p_{w_1}, p_{w_2} \ldots p_{w_m} \ldots p_{w_M}$ of each WiFi node in the area, and selects the maximum power node according to equation (2):

$$p_w = \max\{p_{w_1}, p_{w_2} \ldots p_{w_n} \ldots p_{w_M}\} \tag{2}$$

where the transmitted power p_{w_m} can be expressed as:

$$p_{w_m} = p_{t_m} G_{t_m} G_r (\frac{c}{4\pi f d_m})^2 \tag{3}$$

The formula takes node m as an example, p_{t_w} is the transmit power, G_{t_w} is the transmit gain, G_r is the receive gain, and d_m is the actual distance between the device and the node.

When connecting a WiFi node, the transmission rate r_w of the device is inversely proportional to the number of devices connected to that node, which is obtained from the following equation:

$$r_w = \frac{B_w}{u} lb\left(1 + \frac{p_w \cdot h_w}{\frac{B_w}{u} N_w}\right) \tag{4}$$

where, B_w is the channel bandwidth of the WiFi network, h_w is the channel gain, N_w is the power spectral density of the noise in the channel, and u is the number of devices.

2.2.2. Time delay model. When the device is processing tasks locally, delay D_l contains only the local processing delay, denoted as:

$$D_1 = \frac{c}{F_1} \tag{5}$$

where F_l represents the device CPU arithmetic.

When the device offloads the computation task, the return of the result can be ignored, and the total delay D. Consists of upload delay and processing delay. Therefore, the total delay D_o^{bm} and D_o^{bc} of the computation task offloaded to the MEC server and the cloud computing server via the mobile network can be expressed as follows, respectively:

$$D_o^{bm} = \frac{d}{r_b} + \frac{c}{F_o^m} \tag{6}$$

$$D_o^{bc} = \frac{d}{r_b} + \frac{c}{F_o^c} + t_c \tag{7}$$

The total delay D_o^{wm} and D_o^{we} for the computation task to offload the task to the MEC server and the cloud computing server via the WiFi network can be expressed as follows, respectively:

$$D_o^{wm} = \frac{d}{r_w} + \frac{c}{F_o^m} \tag{8}$$

$$D_o^{wc} = \frac{d}{r_w} + \frac{c}{F_o^c} + t_c \tag{9}$$

where F_o^m and F_o^c are the arithmetic power of the MEC server and the cloud computing server, respectively, and t_e is the queue waiting delay of the cloud computing server.

2.2.3. Energy consumption models. When the device is processing computing tasks locally, energy consumption E_l contains only local processing energy, denoted as:

$$E_1 = z_n \cdot c \tag{10}$$

where z_n is the CPU processing energy consumption, which can be expressed by equation (11):

$$z_n = 10^{-27} \cdot (F_1)^2 \tag{11}$$

Total Energy Consumption for Device Offloading Tasks E. Including transmission energy and waiting energy. The total energy consumption of computing tasks offloaded to MEC servers and cloud computing servers via the mobile network E_o^{bm} and E_o^{bc} can be expressed as follows, respectively:

$$E_o^{bm} = p_b \cdot \frac{d}{r_b} + p_s \cdot \frac{c}{F_o^m} \tag{12}$$

$$E_o^{bc} = p_b \cdot \frac{d}{r_b} + p_s \cdot \left(\frac{c}{F_o^c} + t_c \right) \tag{13}$$

where p_s is the standby power of the device.

The total energy consumption E_o^{wm} and E_o^{wc} for computing tasks offloaded to the MEC server and the cloud server via the WiFi network can be expressed as follows, respectively:

$$E_o^{wm} = p_w \cdot \frac{d}{r_w} + p_s \cdot \frac{c}{F_o^m} \tag{14}$$

$$E_o^{wc} = p_w \cdot \frac{d}{r_w} + p_s \cdot \left(\frac{c}{F_o^c} + t_c \right) \tag{15}$$

2.2.4. Overhead model. During the processing of computational tasks on the device, not only delay and energy consumption are incurred, but also additional system overhead is incurred. It is assumed that no additional overhead is incurred when the task is processed locally at the device, with overhead M_l being 0, and no additional transmission overhead is incurred when the data is uploaded via the WiFi network. Relatively, the mobile network charges a traffic fee based on the data traffic uploaded by the device, and the MEC server and the cloud computing server charge different storage fees and computation fees based on the amount of data and complexity of the offloaded task.

Therefore, the overheads M_o^{bm} and M_o^{bc} for computing tasks to be offloaded to the MEC server and the cloud computing server via the mobile network can be expressed as follows, respectively:

$$M_o^{bm} = M_b \cdot d + M_m^d \cdot d + M_m^c \cdot c \tag{16}$$

$$M_o^{bc} = M_b \cdot d + M_c^d \cdot d + M_c^c \cdot c \tag{17}$$

The overheads M_o^{wm} and M_o^{wc} for offloading the computational tasks to the MEC server and the cloud computing server via the WiFi network are, respectively:

$$M_o^{wm} = M_m^d \cdot d + M_m^c \cdot c \tag{18}$$

$$M_o^{wc} = M_c^d \cdot d + M_c^c \cdot c \tag{19}$$

where M_b is the data traffic overhead coefficient, M_m^d and M_m^c are the storage and computation overhead coefficients of the MEC server, and M_c^d and M_c^c are the storage and computation overhead coefficients of the cloud computing server, respectively.

To summarize, in this paper, the weighted summation of delay, energy consumption, and overhead generated by the above five different task processing modes, corresponding to the total overhead $A_l, A_o^{bm}, A_o^{bc}, A_o^{bc}, A_o^{wc}$ can be expressed as respectively:

$$A_1 = W_d \cdot D_1 + W_e \cdot E_1 + W_m \cdot M_1 \tag{20}$$

$$A_o^{bc} = W_d \cdot D_o^{bc} + W_e \cdot E_o^{bc} + W_m \cdot M_o^{bc} \tag{21}$$

$$A_o^{wm} = W_d \cdot D_o^{wm} + W_e \cdot E_o^{wm} + W_m \cdot M_o^{wm} \tag{22}$$

$$A_o^{wc} = W_d \cdot D_o^{wc} + W_e \cdot E_o^{wc} + W_m \cdot M_o^{wc} \tag{23}$$

$$A_o^{bm} = W_d \cdot D_o^{bm} + W_e \cdot E_o^{bm} + W_m \cdot M_o^{bm} \tag{24}$$

where W_d, W_e, W_m is the weighting factor corresponding to delay, energy consumption and overhead, respectively, obeying $0 \le W_d \le 1, 0 \le W_e \le 1, 0 \le W_m \le 1$ as well as $W_d + W_e + W_m = 1$.

Therefore, the total overhead A_{sum}^t of the devices in the cell to process tasks at discrete moments t can be expressed as:

$$A_{sum}^t = \sum_{n=1}^{N} A_n^t \tag{25}$$

where A_n^t is the total task processing overhead of device n at discrete moments t.

3. Resource optimization scheduling based on DQN algorithm

In this chapter, we will propose an optimization objective for the multiuser task offloading problem in a small area proposed above, model the proposed problem based on reinforcement learning, set the three key elements of reinforcement learning in the optimization process - state, action and reward, and then apply the DQN algorithm in reinforcement learning to solve the optimization problem.

3.1. DQN algorithm

Q-learning is a well-known reinforcement learning (RL) algorithm that has been widely used to solve problems in a variety of domains.The Q-learning algorithm evaluates the advantages and disadvantages of executing an action in a particular state by using an action utility function, Q(S, A). The Q-value of an action a_t performed by an intelligent body in state s_t is denoted by $Q(s_t, a_t)$, where $s_t \in S, a_t \in A$. By continuously interacting with the learning environment and iteratively updating the Q-values of different "state-action" pairs (s_t, a_t) during the training process $Q(s_t, a_t)$, the intelligent body will obtain the optimal action utility function $Q(S, A)$. The Q-value updating method proposed in the Q-learning algorithm is given below:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R(s_t, a_t) + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{26}$$

where $R(s_t, a_t)$ denotes the immediate reward returned to the intelligent body by the system environment after the intelligent body performs an action a_t in state s_t and interacts with the system environment. In addition, $\alpha \in (0,1]$ denotes the learning rate of the intelligent body, which determines how much of this experience is to be learned. $\gamma \in [0,1)$ denotes the discount factor for future rewards, the larger value of which means that the more the intelligent body pays attention to the rewards it may receive in the future.

Based on the optimal action utility function $Q(S, A)$ obtained after training, the optimal decision-making strategy of the intelligent body is as follows:

$$\pi(s) = arg \max_{a \in A}(Q(s, a)), s \in S \tag{27}$$

That is, the optimal decision of the intelligent body in the corresponding state is to choose the action corresponding to the current state that can obtain the maximum Q value.

If the traditional Q-learning algorithm is used, in order to obtain the optimal decision-making strategy, the Q values corresponding to the complete state space and action space need to be calculated. When the state space and action space of the problem to be solved are large, the intelligent body will need to go through a very large training process and the convergence speed of the algorithm is greatly reduced. Therefore, the traditional Q learning algorithm is no longer practical and effective for complex problems. To address this challenge, a deep reinforcement learning algorithm called Deep Q Network (DQN) has been proposed. The DQN algorithm combines the traditional Q learning algorithm with a deep neural network (DNN), where the DNN is used as a function approximation machine to realize an approximation to the utility function of the actions in the Q learning method [30].

The DQN algorithmic framework contains two DNNs: one is called Behavioral Q Network, denoted by $Q(S,A;\theta)$, which is responsible for interacting with the environment and getting interaction samples. One is called the goal Q network, denoted by $Q(S,A;\hat{\theta})$, which is used to generate the goal Q value for training and updating the network. Where, θ and $\hat{\theta}$ denote the parameters of the corresponding network model, respectively. The parameters of the behavioral Q network model θ will be updated in real time during each round of training. And in order to mitigate the model volatility and obtain a smooth model, the parameters $\hat{\theta}$ of the target Q network model will be fixed for a period of time. Whenever a certain number of rounds of iterations of training are completed, the parameters of the behavioral Q network model will be synchronized to the target Q network, and then the next stage of learning will be carried out.

In order to avoid falling into a local optimum, a ò− greedy strategy is used to select actions. Specifically, the intelligent body selects a random action with ò probability, otherwise, the action a_t corresponding to the maximum estimated Q value of the behavioral Q network is selected, i.e:

$$a_t = arg\max_{a\in A} Q(s_t,a;\theta) \tag{28}$$

During the training process, the DQN algorithm adopts an experience playback strategy to make the training effect more stable. First of all, the training system will store the tuple information (s_t,a_t,r_t,s_{t+1}) obtained from the interaction between the intelligent body and the environment in each training round into the storage area u of memory size $u=|u|$. When the storage area is full of training experience, then the latest tuple information each time will overwrite the oldest experience samples in time. At each learning time of the intelligent body, a batch of experience samples (s_j,a_j,r_j,s_{j+1}) will be randomly sampled from the experience storage area for learning, thus reducing the correlation of the training samples and making the model training more stable.

Using the randomly sampled experience samples (s_j,a_j,r_j,s_{j+1}) for learning, the target Q network generates a target Q value of:

$$z_j = r_j + maxQ(s_{j+1},a;\hat{\theta}) \tag{29}$$

The loss function used to train the behavioral Q network is:

$$\Phi(\theta) = (z_j - Q(s_j,a_j;\theta))^2 \tag{30}$$

Parameter θ of the behavioral Q-network model is updated by performing gradient descent to minimize the loss.

3.2. Optimization objectives

The previous section weighted and summed the latency, energy consumption, and overhead incurred by a device's computational task processing to obtain the weighted total overhead $A_n^{'}$ of a single device processing a task at discrete time point t. From a global perspective, this section sets the

optimization objective to minimize the total task processing overhead A_{sum}^{T} of all the devices of the system within a limited time $0-T$, which can be expressed by equation (31):

$$\min A_{sum}^{T} = \min \sum_{t=1}^{T} \sum_{n=1}^{N} A_n^t \tag{31}$$

where N is the number of cell devices and T is the cutoff moment.

3.3. Problem modeling

In order to use the DQN algorithm to solve a device with an offload request for a computational task (IRD), we first model the problem as an MDP problem, i.e., we define the state space S^O, the action space A^O, and the reward function R^O of the problem.

1) State. In order for the DQN intelligences to learn the optimal decisions and minimize the total system overhead, the state information obtained by the intelligences observing the system should include the offloading strategy for better performance. Therefore the state $s_t^O \in S^O$ obtained by the DQN intelligent body observing the system in time slot t is:

$$s_t^O = \{L_t, V_t, D_t, h_t, \hat{h}_t\} \tag{32}$$

where, L_t is the data size information for all IRD device computation tasks, i.e., $L_t = \{L_1(t), L_2(t), \ldots, L_M(t)\}$. V_t is the demand computation information for all IRD device computation tasks, i.e., $V_t = \{V_1(t), V_2(t), \ldots, V_M(t)\}$. D_t is the maximum delay information for all IRD device computation tasks, i.e., $D_t = \{D_1(t), D_2(t), \ldots, D_M(t)\}$. h_t is the channel gain information from all IRD devices to the base station on all sub-channels, i.e., $h_t = \{h_{m,n}(t), \forall m \in \mathrm{M}, n \in \mathrm{N}\}$. $\hat{h}_t$ is the channel gain information on all sub-channels from all IRD devices to their corresponding devices with sufficient computing power, i.e., $\hat{h}_t = \{\hat{h}_{m,n}(t), \forall m \in \mathrm{M}, n \in \mathrm{N}\}$.

2) Action. The output of the DQN intelligence should be the offloading policy to be solved, so the action a_t^O output by the DQN intelligence at time slot t based on the state s_t^O obtained from the observation is:

$$a_t^O = \{o_t, l_t, f_t, x_t\} \tag{33}$$

Also note that the MEC server computational resources have been discretized into CN computational resource blocks CRBs in order to make the output discretization consistent with the requirements of the DQN algorithm, so the computational resource allocation for IRD device m in f_t:

$$f_m(t) \in \left\{ \frac{1}{CN} F^{MEC}, \frac{2}{CN} F^{MEC}, \ldots, F^{MEC} \right\} \tag{34}$$

The DQN intelligent body outputs an action a_t^O at time slot t based on the state s_t^O obtained from observation, and after executing the action it receives an immediate reward r_t^O from the system and the state is transferred to s_{t+1}^O.

3) Reward. We use the DQN algorithm to solve the solved offloading policy objective is to minimize the total system overhead, while the DQN algorithm maximizes the reward through continuous learning, so the total system overhead part of the reward function should take the opposite number. In addition, for actions that do not satisfy the constraints of the modeling problem, an additional certain negative value should be given as a penalty to guide the DQN intelligences to avoid outputting these decisions.

Also, in order to incentivize the DQN intelligences to learn decisions with smaller total system overhead as well as to increase the impact of these decisions on the network parameter updates, we consider giving an additional positive reward to the intelligences if the total system overhead computed from the actions outputted by the DQN intelligences is smaller compared to that of the baseline scenario.The DQN intelligences are rewarded with an additional positive reward for the actions they have outputted after outputting the actions a_t^O based on the states obtained from the observations s_t^O The Immediate reward is defined as:

$$r_t^O = \begin{cases} -\sum_{m=1}^{M} U_m + 2, \text{Satisfy the constraint}, \sum_{m=1}^{M} U_m < U^{base} \\ -\sum_{m=1}^{M} U_m, \text{Satisfy the constraint}, \sum_{m=1}^{M} U_m \geq U^{base} \\ -\sum_{m=1}^{M} U_m - 2, \text{Constraints are not met} \end{cases} \tag{35}$$

3.4. Optimization problem solving

The state space, action space and reward function of the offloading strategy have been defined in the previous section, and the DQN algorithm is utilized to train the DQN intelligences with the input of a given state. The offloading strategy sought in this paper is obtained by iteratively solving the problem using the designed algorithm, where the total energy consumption of the system decreases in each iteration, and eventually convergence is reached to obtain a suboptimal solution to the problem.

4. Simulation results and analysis

4.1. Experimental environment and setup

In this paper, the proposed scheme is evaluated through simulation experiments, in which the hardware configuration is: Intel i7 CPU, 8G RAM, 512M SSD and 1T mechanical hard disk. The experimental operating system is Windows 10 Professional, the development language is Java language, and the development platform is Eclipse platform.

The experiment assumes that there are a total of five factories in the scenario, a total of 100 industrial equipment, the number of heterogeneous edge servers is 15, two-thirds of the edge servers are distributed in the five factories, and the other one-third is deployed in the central server room. Among them, the devices under the same factory can interact with each other through D2D technology to realize tasks, and the edge servers can communicate with each other through P2P technology to realize resource sharing. Assuming that the resources that the cloud servers can provide are unlimited, tasks offloaded to the cloud do not need to be queued and can be processed upon arrival.

The experiments are set up with task sizes between [50,500] in increments of 50, and the data sizes of all computation tasks are randomly distributed in the range of [200,3000] KB. The number of CPU cycles required by a computational task is related to its data size. Industrial devices have limited resources and computing power, so this paper specifies that their computing power is randomly distributed at [0.5,1] Gcycles/bit. The remote cloud server has a computing power of 6 Gcycles/bit. The computing power of the edge servers should be in between the devices and the cloud servers, and after comprehensive consideration, the experiment allocates computing power to the edge servers in the range of [1,5] Gcycles/bit.

4.2. Experimental Comparison Subjects

1) First-come-first-served policy (S1): the DQR algorithm is used as the base algorithm and the scheduling process uses the first-come-first-served rule.

2) Later-first service policy (S2): the DQR algorithm is used as the base algorithm and the scheduling process uses the later-first service rule.

3) Randomized service policy (S3): the DQR algorithm is used as the base algorithm and the scheduling process uses the randomized service rule.

4.3. Experimental results and analysis

4.3.1. Convergence Performance Evaluation. In this paper, the convergence performance of the reinforcement learning algorithm is evaluated, and the variation of the loss function of the DQR algorithm during the training process is shown in Fig. 2. In the initial stage of training, the value of the loss function rises. This is because the small size of the data does not represent the overall data distribution, so after a small amount of data training, the value of the loss function shows an upward trend after subsequent data inputs. Afterwards, as the number of training steps increases, after more data training, the performance of the network will gradually improve, and the value of the loss function will gradually decrease, and finally converge to a relatively stable value. In addition, it can be observed that the proposed algorithm converges in about 9000 steps, which also indicates that the method proposed in this paper has good convergence performance.

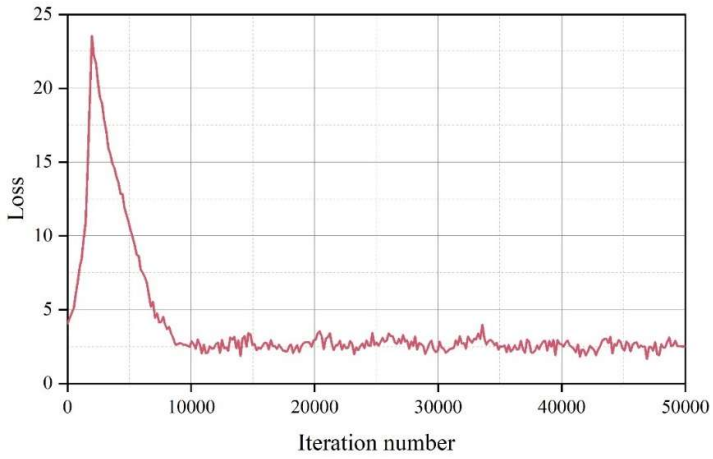


Fig. 2. The change in the loss function in the training process

The cumulative rewards of the training process for different service offloading strategies are shown in Figure 3. In order to avoid the interference of random values, the data used in this paper is the average of 50 independent experiments. The poor performance in the early part of the curve is due to the fact that the reinforcement learning algorithm uses a randomized strategy to collect environmental information, so the early part of the curve is mainly in the exploratory stage and gradually adapts to the environment. After completing the environmental information collection and training, the computational offloading strategy based on the DQR algorithm proposed in this paper starts to improve the average reward after 20 episodes and basically converges in the 30th episodes, and the long-term average reward is basically stable thereafter. In contrast, the convergence speeds of the other strategies are not satisfactory.The S3 strategy has no obvious convergence trend after 200 episodes.The S1 strategy has better convergence than the S2 strategy and converges after the 40th episodes, but its long-term average reward is lower than that of the offloading strategy based on the DQR algorithm proposed in this paper.

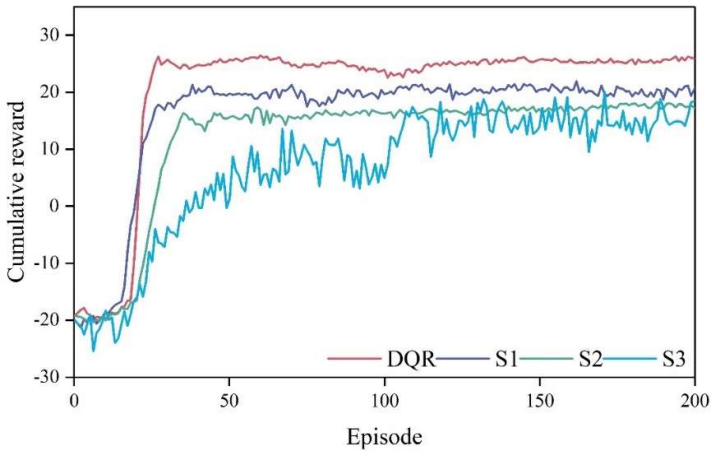


Fig. 3. Long-term average rewards for different service discharge strategies

4.3.2. Integrated performance assessment:

1) Task Completion. The effectiveness of the offloading strategy based on the DQR algorithm can be demonstrated by comparison experiments with other common strategies under the same conditions. In this section, the offloading strategy based on DQR algorithm is compared with first-come-first-served rule, later-first-served rule, randomized service rule, etc., on the index of task completion in individual application mode, and the comparison results on task completion are shown in Fig. 4. It can be seen that overall the offloading strategy based on DQR algorithm is better than other strategies. The size of the overall task completion of different strategies at different scales spans a large range, while the size of different strategies at the same scale spans a relatively small range, a situation that causes the other strategies to look basically the same.

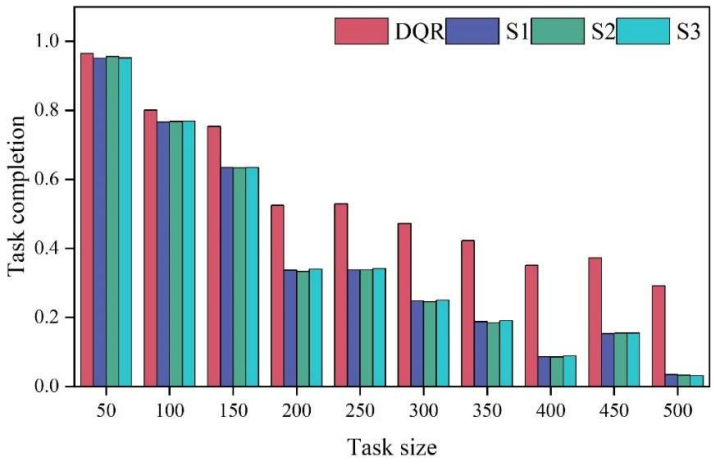
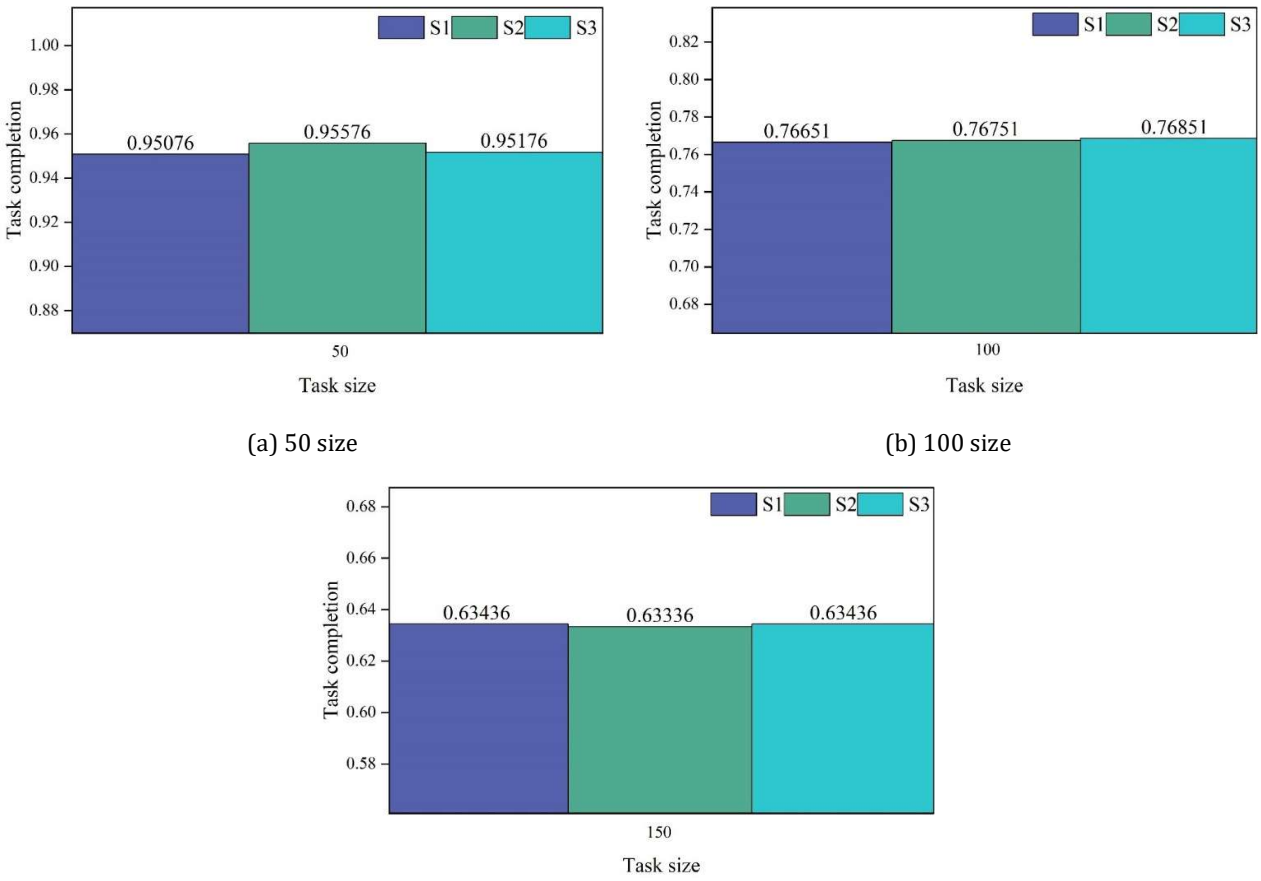


Fig. 4. Comparison results of task completion

In order to see the different performances of the other three strategies more graphically, the actual results of S1, S2, and S3 under three task sizes are intercepted respectively, and the task completion degrees are shown in Fig. 5. (a), (b), and (c) represent the task completion degree of the three types of strategies under the task sizes of 50, 100, and 150, respectively. The comparison of the task completion degree of these three types of strategies does not have a fixed size relationship, there are high and low, which indicates that for this scenario experimented in this paper, the three strategies compared can play a basically similar role. The task completion of the offloading strategy based on the DQR algorithm is 0.96463, 0.80021, and 0.75314 for task sizes of 50, 100, and 150, respectively, which are higher than those of the three types of strategies.



(a) 50 size

(b) 100 size

(c) 150 size

Fig. 5. Task completion on different tasks

2) Task completion time. The comparison results regarding task completion time are shown in Fig. 6. The task completion time of the offloading strategy based on the DQR algorithm is reduced by 7.93%, 14.91%, and 13.72% compared to S1, S2, and S3, respectively.

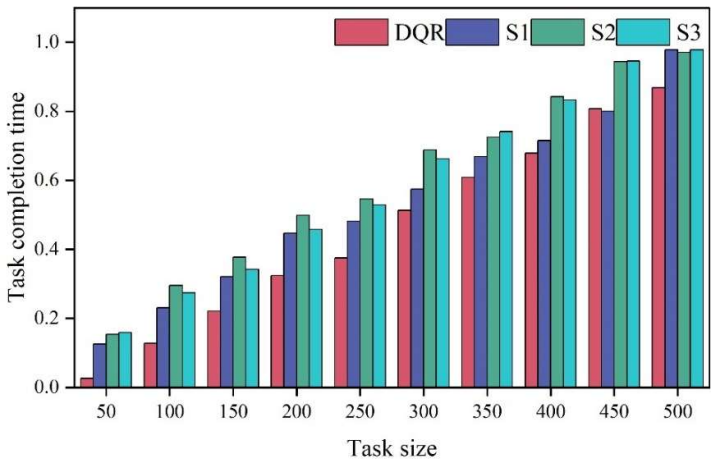


Fig. 6. The comparison of the task completion time

3) Algorithm running time. The comparison results about the algorithm running time are shown in Fig. 7. The algorithmic running time of the offloading strategy based on the DQR algorithm is improved by 7.51%, 3.58%, and 17.95%, respectively, compared with the other strategies. This is because the

offloading strategy based on the DQR algorithm needs to consider the update of all individuals within the population during iteration, so it takes much longer than the other strategies compared.

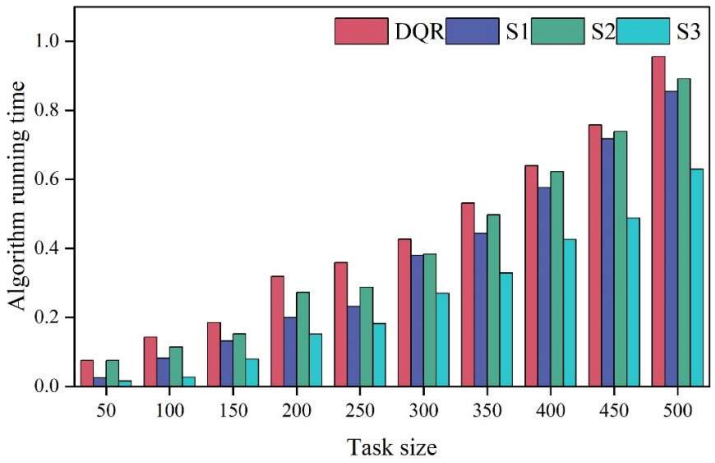


Fig. 7. The comparison of the algorithm running time

4) Task processing cost. The comparison results about the task processing cost are shown in Fig. 8. The task processing cost of the offloading strategy based on the DQR algorithm is reduced by 36.57%, 41.7%, and 45.55% compared to S1, S2, and S3, respectively.

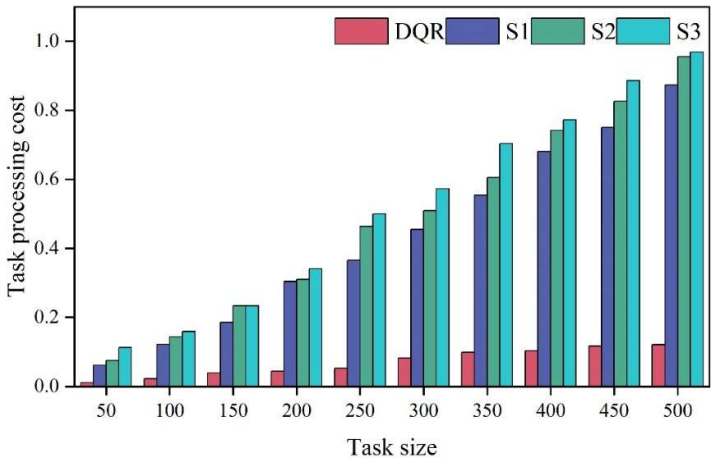


Fig. 8. The comparison of the task processing cost

In summary, the offloading strategy based on the DQN algorithm finally achieves the optimal scheduling of industrial Internet resources with the goal of minimizing the total task processing overhead.

5. Conclusion

In this paper, we propose a data computing task offloading system architecture for industrial internet, under which we establish a system computing resource scheduling model, construct the optimization problem as MDP with the optimization objectives of reducing equipment energy consumption and system computing overhead, and study its optimal scheduling strategy based on reinforcement learning. The offloading strategy based on DQN algorithm reduces the task processing cost by 36.57%, 41.7%, and 45.55%, and the task completion time by 7.93%, 14.91%, and 13.72%, respectively, compared with the first-come-first-served rule, later-first-served rule, and random service rule

strategies. It shows that the offloading strategy based on DQN algorithm has a better optimization of device energy consumption and system computational overhead.

References

[1] Ascorti, L., Savazzi, S., Soatti, G., Nicoli, M., Sisinni, E., & Galimberti, S. (2017). A wireless cloud network platform for industrial process automation: Critical data publishing and distributed sensing. IEEE Transactions on Instrumentation and Measurement, 66(4), 592-603.

[2] Wang, J., Xu, C., Zhang, J., Bao, J., & Zhong, R. (2020). A collaborative architecture of the industrial internet platform for manufacturing systems. Robotics and Computer-Integrated Manufacturing, 61, 101854.

[3] Jamil, M. N., Schelén, O., Monrat, A. A., & Andersson, K. (2024). Enabling Industrial Internet of Things by Leveraging Distributed Edge-to-Cloud Computing: Challenges and Opportunities. IEEE Access.

[4] Liang, F., Yu, W., Liu, X., Griffith, D., & Golmie, N. (2020). Toward computing resource reservation scheduling in Industrial Internet of Things. IEEE Internet of Things Journal, 8(10), 8210-8222.

[5] Luo, Q., Hu, S., Li, C., Li, G., & Shi, W. (2021). Resource scheduling in edge computing: A survey. IEEE Communications Surveys & Tutorials, 23(4), 2131-2165.

[6] Kaur, K., Garg, S., Aujla, G. S., Kumar, N., Rodrigues, J. J., & Guizani, M. (2018). Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay. IEEE communications magazine, 56(2), 44-51.

[7] Hong, Z., Chen, W., Huang, H., Guo, S., & Zheng, Z. (2019). Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments. IEEE transactions on parallel and distributed systems, 30(12), 2759-2774.

[8] Xie, R., Gu, D., Tang, Q., Huang, T., & Yu, F. R. (2022). Workflow scheduling in serverless edge computing for the industrial internet of things: A learning approach. IEEE Transactions on Industrial Informatics, 19(7), 8242-8252.

[9] Wang, K., Zhou, Y., Liu, Z., Shao, Z., Luo, X., & Yang, Y. (2020). Online task scheduling and resource allocation for intelligent NOMA-based industrial Internet of Things. IEEE Journal on Selected Areas in Communications, 38(5), 803-815.

[10] Xia, W., Zhang, J., Quek, T. Q., Jin, S., & Zhu, H. (2020). Mobile edge cloud-based industrial Internet of Things: Improving edge intelligence with hierarchical SDN controllers. IEEE Vehicular Technology Magazine, 15(1), 36-45.

[11] Wang, Y., Yang, S., Ren, X., Zhao, P., Zhao, C., & Yang, X. (2021). IndustEdge: A time-sensitive networking enabled edge-cloud collaborative intelligent platform for smart industry. IEEE Transactions on Industrial Informatics, 18(4), 2386-2398.

[12] Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M., & Wu, D. O. (2020). Edge computing in industrial internet of things: Architecture, advances and challenges. IEEE Communications Surveys & Tutorials, 22(4), 2462-2488.

[13] Song, X., Wang, Y., Xie, Z., & Xia, L. (2021). A Cloud-Edge Collaborative Computing Task Scheduling and Resource Allocation Algorithm for Energy Internet Environment. KSII Transactions on Internet and Information Systems (TIIS), 15(6), 2282-2303.

[14] Chen, Y., Liu, Z., Zhang, Y., Wu, Y., Chen, X., & Zhao, L. (2020). Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things. IEEE Transactions on Industrial Informatics, 17(7), 4925-4934.

[15] Hou, W., Wen, H., Zhang, N., Wu, J., Lei, W., & Zhao, R. (2021). Incentive-driven task allocation for collaborative edge computing in industrial internet of things. IEEE Internet of Things Journal, 9(1), 706-718.

[16] Li, X., Wan, J., Dai, H. N., Imran, M., Xia, M., & Celesti, A. (2019). A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing. IEEE Transactions on Industrial Informatics, 15(7), 4225-4234.

[17] Amer, A. A., Talkhan, I. E., Ahmed, R., & Ismail, T. (2022). An optimized collaborative scheduling algorithm for prioritized tasks with shared resources in mobile-edge and cloud computing systems. Mobile Networks and Applications, 27(4), 1444-1460.

[18] Ma, S., Huang, Y., Chen, Y., Xiao, Q., Xu, J., & Leng, J. (2024). Edge-Cloud Cooperation Driven Intelligent Sustainability Evaluation Strategy Based on IoT and CPS for Energy-Intensive Manufacturing Industries. IEEE Internet of Things Journal.

[19] Qi, Y., Pan, L., & Liu, S. (2024). Service provisioning based on edge-cloud collaboration: A two-timescale online scheduling algorithm. IEEE Internet of Things Journal.

[20] Sun, Z., Yang, H., Li, C., Yao, Q., Wang, D., Zhang, J., & Vasilakos, A. V. (2021). Cloud-edge collaboration in industrial internet of things: A joint offloading scheme based on resource prediction. IEEE Internet of Things Journal, 9(18), 17014-17025.

[21] Yang, C., Liao, F., Lan, S., Wang, L., Shen, W., & Huang, G. Q. (2023). Flexible resource scheduling for software-defined cloud manufacturing with edge computing. Engineering, 22, 60-70.

[22] Okwuibe, J., Haavisto, J., Kovacevic, I., Harjula, E., Ahmad, I., Islam, J., & Ylianttila, M. (2021). SDN-enabled resource orchestration for industrial IoT in collaborative edge-cloud networks. IEEE Access, 9, 115839-115854.

[23] Laili, Y., Guo, F., Ren, L., Li, X., Li, Y., & Zhang, L. (2021). Parallel scheduling of large-scale tasks for industrial cloud–edge collaboration. IEEE Internet of Things Journal, 10(4), 3231-3242.

[24] Wang, Y., Fang, J., Cheng, Y., She, H., Guo, Y., & Zheng, G. (2023). Cooperative End-Edge-Cloud Computing and Resource Allocation for Digital Twin Enabled 6G Industrial IoT. IEEE Journal of Selected Topics in Signal Processing.

[25] Ma, J., Zhou, H., Liu, C., Mingcheng, E., Jiang, Z., & Wang, Q. (2020). Study on edge-cloud collaborative production scheduling based on enterprises with multi-factory. IEEE Access, 8, 30069-30080.

[26] Zhang, X., Wang, M., Zhu, X., Yan, Z., & Geng, G. (2025). Collaborative Edge-Cloud Data Transfer Optimization for Industrial Internet of Things. IEEE Transactions on Parallel and Distributed Systems.

[27] Avval Danial Bakhshayeshi,Heris Pouria Ouni,Navimipour Nima Jafari,Mohammadi Behnaz & Yalcin Senay. (2022). A new QoS-aware method for production scheduling in the industrial internet of things using elephant herding optimization algorithm. Cluster Computing(6),3611-3626.

[28] Souri Alireza,Norouzi Monire & Alsenani Yousef. (2023). A new cloud-based cyber-attack detection architecture for hyper-automation process in industrial internet of things. Cluster Computing(3),3639-3655.

[29] Wuhui Chen,Zhen Zhang,Zicong Hong,Chuan Chen,Jiajing Wu,Sabita Maharjan... & Yan Zhang 0002. (2019). Cooperative and Distributed Computation Offloading for Blockchain-Empowered Industrial Internet of Things. IEEE Internet of Things Journal(5),8433-8446.

[30] Fan Liang,Wei Yu,Xing Liu,David Griffith & Nada Golmie. (2022). Towards Deep Q-Network Based Resource Allocation in Industrial Internet of Things. IEEE internet of things journal(12).