Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

# Multi-agent deep reinforcement learning for end–edge orchestrated resource allocation in industrial wireless networks*

Xiaoyu LIU[1,2,3,4], Chi XU‡[1,2,3], Haibin YU‡[1,2,3], Peng ZENG[1,2,3]

[1]*State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China*

[2]*Key Laboratory of Networked Control Systems, Chinese Academy of Sciences, Shenyang 110016, China*

[3]*Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang 110169, China*

[4]*University of Chinese Academy of Sciences, Beijing 100049, China*

E-mail: liuxiaoyu1@sia.cn; xuchi@sia.cn; yhb@sia.cn; zp@sia.cn

Received July 5, 2021; Revision accepted Nov. 15, 2021; Crosschecked Dec. 31, 2021

**Abstract:** Edge artificial intelligence will empower the ever simple industrial wireless networks (IWNs) supporting complex and dynamic tasks by collaboratively exploiting the computation and communication resources of both machine-type devices (MTDs) and edge servers. In this paper, we propose a multi-agent deep reinforcement learning based resource allocation (MADRL-RA) algorithm for end–edge orchestrated IWNs to support computation-intensive and delay-sensitive applications. First, we present the system model of IWNs, wherein each MTD is regarded as a self-learning agent. Then, we apply the Markov decision process to formulate a minimum system overhead problem with joint optimization of delay and energy consumption. Next, we employ MADRL to defeat the explosive state space and learn an effective resource allocation policy with respect to computing decision, computation capacity, and transmission power. To break the time correlation of training data while accelerating the learning process of MADRL-RA, we design a weighted experience replay to store and sample experiences categorically. Furthermore, we propose a step-by-step $\varepsilon$-greedy method to balance exploitation and exploration. Finally, we verify the effectiveness of MADRL-RA by comparing it with some benchmark algorithms in many experiments, showing that MADRL-RA converges quickly and learns an effective resource allocation policy achieving the minimum system overhead.

**Key words:** Multi-agent deep reinforcement learning; End–edge orchestrated; Industrial wireless networks; Delay; Energy consumption

https://doi.org/10.1631/FITEE.2100331 **CLC number:** TN929.5; TP18

## 1 Introduction

With the rapid development of intelligent manufacturing, massive distributed machine-type devices (MTDs) interconnected by industrial wireless networks (IWNs) generate a vast amount of heterogeneous data, which are generally computation-intensive and delay-sensitive (Yao et al., 2019; Xu et al., 2021; Yu et al., 2021). For resource-constrained MTDs, it is a challenge to process

‡ Corresponding authors

* Project supported by the National Key R&D Program of China (No. 2020YFB1710900), the National Natural Science Foundation of China (Nos. 62173322, 61803368, and U1908212), the China Postdoctoral Science Foundation (No. 2019M661156), and the Youth Innovation Promotion Association, Chinese Academy of Sciences (No. 2019202)

ORCID: Xiaoyu LIU, https://orcid.org/0000-0003-3293-0803; Chi XU, https://orcid.org/0000-0001-7389-5763; Haibin YU, https://orcid.org/0000-0002-1663-2956

data locally. Thus, a centralized cloud computing paradigm was first proposed (Kumar et al., 2019). The cloud server acts as a centralized intelligent agent to uniformly schedule computation resources for MTDs. Correspondingly, MTDs offload data to the cloud server and download the processed data. However, owing to the nature of centralized services, the cloud server is typically deployed far from MTDs. Therefore, offloading data from MTDs to the cloud server triggers significant communication delay, which is intolerable for real-time applications, such as robot control, augmented/virtual reality, and automatic drive.

To address the above issues associated with cloud computing, the multi-access edge computing (MEC) paradigm was proposed (Porambage et al., 2018), which distributes computation resources to massive edge servers deployed in base stations, access points, and other edge infrastructures. However, the high concurrent offloading of massive MTDs may result in traffic congestion and edge server overload, adding additional delay and energy consumption. Meanwhile, during the manufacturing process, it is a serious challenge for energy-constrained MTDs that MTDs generate and offload data continuously. Thus, to make full use of the computation resources of MTDs and edge servers while extending the energy lifetime of MTDs, a flexible resource allocation policy for integrating the resources of MTDs and edge servers is urgently required (Zhang YM et al., 2020; Ren et al., 2021). With end–edge orchestrated resource allocation, IWNs can support massive computation-intensive and delay-sensitive industrial applications simultaneously.

Generally, conventional model-driven algorithms require complete system information to establish an accurate system model and obtain an effective resource allocation policy. However, the dynamic and time-varying nature of end–edge orchestrated IWNs renders it difficult to collect the complete system information for establishing an accurate system model. On the contrary, reinforcement learning (RL), as a self-learning artificial intelligence (AI) technology, uses agents interacting with IWNs to obtain local system information and approximate the system model (Shakarami et al., 2020; Wang et al., 2020). However, the local system information from distributed MTDs is numerous and coupled, resulting in an explosive state space. To tackle the state space explosion, deep learning (DL) is used to express the relationship of local system information by interconnected weighted neurons. Thus, combining DL and RL, i.e., using deep reinforcement learning (DRL), to obtain the resource allocation policy not only gives play to the self-learning advantage of RL but also uses DL to deal with state space explosion.

Previous works (Lin et al., 2019; Wei et al., 2019; Liu KH and Liao, 2020; Lu et al., 2020; Xiong et al., 2020; Chen et al., 2021; He et al., 2021; Liu XY et al., 2021) used DRL to optimize resource allocation in wireless networks mainly from the perspective of centralized intelligence. In other words, a single agent collects the global system information to approximate the system model and learn the resource allocation policy. However, in end–edge orchestrated IWNs, distributed MTDs are mobile, and the available resources of MTDs and edge servers are time-varying. In this case, it is difficult for a single agent to track the global system information. Besides, the delay and energy consumption during the collection of global system information may be intolerable for real-time applications. On the contrary, distributed MTDs have potential advantages to achieve swarm intelligence. Every MTD acts as an independent and intelligence-endogenous (Zhang P et al., 2022) agent that can easily observe its local system information, and the cooperation among multiple MTDs (i.e., multiple agents) (Zhang KQ et al., 2021) can approximate the system model, achieve logical resource allocation, and adapt to the dynamic end–edge orchestrated IWNs.

Fully considering the advantages of multiple agents, we use multi-agent DRL (MADRL) to solve the resource allocation problem for end–edge orchestrated IWNs. In this work, we first establish the system model of end–edge orchestrated IWNs. Then, we formulate the minimum system overhead with joint optimization of delay and energy consumption, and apply the Markov decision process (MDP) to express it. Next, we propose an MADRL based resource allocation (MADRL-RA) algorithm, wherein the algorithm architecture, learning process, and computational complexity are presented in detail. Finally, in comparison with some benchmark algorithms, we verify the performance of MADRL-RA by many experiments.

The main contributions are summarized as follows:

1. Considering the diverse constraints on resource allocation, we apply an MDP to formulate the joint optimization of delay and energy consumption, and use MADRL to learn an effective resource allocation policy for minimizing the system overhead with respect to delay and energy consumption.

2. To ensure that the training data are independent and identically distributed while accelerating the learning process of MADRL-RA, we design a weighted experience replay to store and sample experiences categorically.

3. To balance the exploration and exploitation of knowledge about IWNs, we propose a step-by-step $\varepsilon$-greedy method to adjust the probabilities of exploration and exploitation dynamically.

## 2 Related works

For optimizing resource allocation in wireless networks, there are many non-AI algorithms (Guo et al., 2017; Tang and He, 2018; Zhang GL et al., 2018; Feng et al., 2019; Li et al., 2020) considering the joint optimization objects of delay and energy consumption. However, the accurate system model must be known in the above non-AI works, which is difficult for dynamic and time-varying end–edge orchestrated IWNs. Thus, we focus on DRL-based algorithms including single- and multi-agent DRL algorithms.

### 2.1 Single-agent DRL algorithms

For single-agent DRL algorithms, the base station usually acts as a single agent to optimize resource allocation. Previous works (Lin et al., 2019; Xiong et al., 2020) proposed improved value-based deep $Q$ network (DQN) algorithms to efficiently exploit the computation capacities of edge servers and significantly reduce the average delay and energy consumption. Similarly, Alfakih et al. (2020) proposed a value-based SARSA algorithm to deal with the issue of resource management and make an optimal offloading decision to minimize the system cost of energy consumption and computing delay. Several works (Lu et al., 2020; Chen et al., 2021; He et al., 2021) proposed improved deep deterministic policy gradient (DDPG) algorithms to optimize the service latency, energy consumption, and task success rate. In addition, several works (Wei et al., 2019; Liu KH and Liao, 2020) presented actor–critic-based algorithms to minimize the delay and energy consumption and optimize caching and channel access.

### 2.2 Multi-agent DRL algorithms

Mobile MTDs and time-varying resources result in the dynamic and time-varying nature of IWNs. It is difficult for single-agent DRL algorithms to tackle the dynamic and time-varying nature and learn an optimal policy. Therefore, some multi-agent DRL algorithms (Foerster et al., 2016; Lowe et al., 2017; Rashid et al., 2018) have been proposed to solve the above problem, wherein distributed users act as agents. By cooperation among distributed agents, agents observe the local system information to approximate the system model and learn the resource allocation policy. There have been several works applying multi-agent DRL algorithms in industry. For example, Cao et al. (2020) used a multi-agent DDPG algorithm to design multi-channel access and offloading control in Industry 4.0, and Zhu et al. (2021) used the multi-agent DDPG algorithm to minimize the total task processing delay in the multi-vehicle environment. Chu et al. (2020) applied a multi-agent advantage actor–critic algorithm to adapt traffic signal control in complex transportation networks.

## 3 System model

### 3.1 Network model

As shown in Fig. 1, we consider an end–edge orchestrated IWN with $N$ industrial base stations (IBSs) and $M$ MTDs. All distributed IBSs constitute a set of IBSs $\mathcal{N} = \{1, 2, \cdots, N\}$, and all distributed MTDs constitute a set of MTDs $\mathcal{M} = \{1, 2, \cdots, M\}$. Each IBS is equipped with an edge server. MTDs are randomly distributed and generate heterogeneous data. Generally, these data are different in data size and required computation resource. We denote the data of the $m^{\text{th}}$ MTD as $D_m = \{d_m, c_m\}$, where $d_m$ and $c_m$ denote the data size and required computation resource, respectively. Meanwhile, we denote the transmission power of the $m^{\text{th}}$ MTD as $p_m$, where $p_m \in [0, P]$. Specifically, 0 denotes the zero transmission power, and $P$ denotes the maximum transmission power.

Furthermore, the coverage radius of the $n^{\text{th}}$ IBS is denoted as $\mu_n$. If the distance between the $m^{\text{th}}$ MTD and the $n^{\text{th}}$ IBS (denoted as $l_m^n$) is less than

$\mu_n$, the $m^{\text{th}}$ MTD can communicate with the $n^{\text{th}}$ IBS. At any time-slot, each MTD can communicate with only one IBS. The communication rate between MTDs and IBSs varies depending on the bandwidth, transmission power, noise interference, and inter-cell interference. In detail, the communication rate between the $m^{\text{th}}$ MTD and the $n^{\text{th}}$ IBS can be expressed as

$$x_m^n = W_n \log_2\left(1+\frac{p_m h_m^n \left(l_m^n\right)^{-\alpha}}{\sigma^2+\sum\limits_{m'\in\mathcal{M},n'\in\mathcal{N}} p_{m'} h_{m'}^{n'} \left(l_{m'}^{n'}\right)^{-\alpha}}\right), \tag{1}$$

where $W_n$ denotes the bandwidth, $p_m$ the transmission power, $h_m^n$ the channel gain, $l_m^n$ the link distance, $\alpha$ the path loss exponent, $\sigma^2$ the noise power, $m'$ and $n'$ the MTDs and IBSs other than the $m^{\text{th}}$ MTD and the $n^{\text{th}}$ IBS respectively, and $p_{m'} h_{m'}^{n'} \left(l_{m'}^{n'}\right)^{-\alpha}$ the inter-cell interference.

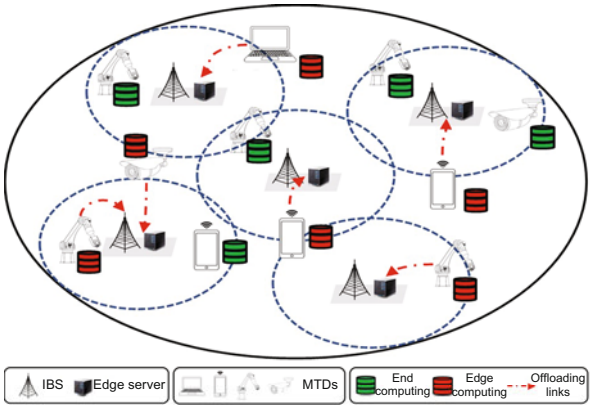


**Fig. 1 Network model (IBS: industrial base station; MTD: machine-type device)**

## 3.2 Computation model

During manufacturing processes, MTDs can make flexible computing decisions to process data, including end computing and edge computing. Specifically, we adopt $o_m^j$ to denote the computing decision, where $o_m^j \in \{0,1\}$ and $j \in \{0,1,...,N\}$. In detail, $o_m^0 = 1$ indicates that data $D_m$ are processed locally (i.e., end computing), and $o_m^n = 1$ indicates that data $D_m$ are offloaded to the $n^{\text{th}}$ IBS (i.e., edge computing). Note that each MTD chooses only one computing decision satisfying the following constraint:

$$\sum_{j=0}^{N} o_m^j = 1. \tag{2}$$

In both end computing and edge computing, MTDs consume time and energy. Next, we express the delay and energy consumption of end computing and edge computing.

### 3.2.1 End computing

If the $m^{\text{th}}$ MTD decides to process data locally, the entire local computation capacity will be used to process data (Lu et al., 2020). Hence, in end computing, the delay includes only the processing delay, which is expressed as

$$T_m^{\text{l}} = \frac{c_m}{f_m^{\text{l}}}, \tag{3}$$

where $f_m^{\text{l}}$ denotes the local computation capacity of the $m^{\text{th}}$ MTD.

In end computing, the energy consumption also includes only that for data processing, which is expressed as

$$E_m^{\text{l}} = \eta c_m \left(f_m^{\text{l}}\right)^2, \tag{4}$$

where $\eta$ denotes the valid switched capacitance related to the chip architecture (Dai et al., 2020).

### 3.2.2 Edge computing

If the $m^{\text{th}}$ MTD decides to offload data to the $n^{\text{th}}$ IBS, the data need to be transmitted via uplink networks. After processing the data in the $n^{\text{th}}$ IBS, the $m^{\text{th}}$ MTD downloads the processed data from the $n^{\text{th}}$ IBS via downlink networks. Because the size of the processed data is much smaller than $d_m$, the downlink delay and downlink energy consumption are very low. Hence, in edge computing, we ignore the downlink transmission process.

In edge computing, the delay includes the uplink transmission delay and processing delay, expressed as

$$T_m^n = \frac{d_m}{x_m^n} + \frac{c_m}{f_m^n}, \tag{5}$$

where $f_m^n$ denotes the computation capacity assigned to the $m^{\text{th}}$ MTD by the edge server located at the $n^{\text{th}}$ IBS.

In edge computing, the energy consumption includes that for both uplink transmission and processing, expressed as

$$E_m^n = \frac{p_m d_m}{x_m^n} + c_m e_n, \tag{6}$$

where $e_n$ denotes the energy consumption per unit of computation capacity of the edge server located at the $n^{\text{th}}$ IBS (Dai et al., 2020).

Specifically, in edge computing, several MTDs may be associated with the same edge server simultaneously. Each MTD is solely assigned to a part of the computation capacity of the edge server. Therefore, $f_m^n$ should satisfy the following constraint:

$$\sum_{m\in\mathcal{M},n\in\mathcal{N}} o_m^n f_m^n \le F_n, \tag{7}$$

where $F_n$ denotes the entire computation capacity of the edge server located at the $n^{\text{th}}$ IBS.

## 4 Problem formulation

### 4.1 Joint optimization problem

To make full use of the computation capacity while reducing the energy consumption, we optimize the weighted sum of delay and energy consumption, namely the system overhead. The joint optimization problem is formulated as follows:

$$\begin{aligned}
\min_{o_m^j,p_m}\ & \sum_{m\in\mathcal{M},j\in\{0,\mathcal{N}\}} o_m^j\left(\omega T_m+(1-\omega)E_m\right)\\
\text{s.t. C1}:\ & 0\le p_m\le P,\\
\text{C2}:\ & \sum_{m\in\mathcal{M},n\in\mathcal{N}} o_m^n f_m^n\le F_n,\\
\text{C3}:\ & 0\le f_m^n\le F_n,\\
\text{C4}:\ & \sum_{j=0}^{N} o_m^j=1,\\
\text{C5}:\ & o_m^j\in\{0,1\},
\end{aligned} \tag{8}$$

where $\omega$ denotes the weight factor of delay, $1-\omega$ the weight factor of energy consumption, $T_m$ the actual delay, and $E_m$ the actual energy consumption. Specifically, if the $m^{\text{th}}$ MTD processes data in end computing, $T_m=T_m^{\text{l}}$ and $E_m=E_m^{\text{l}}$. Otherwise, if the $m^{\text{th}}$ MTD processes data in edge computing, $T_m=T_m^n$ and $E_m=E_m^n$.

Among these constraints, C1 ensures that the transmission power cannot exceed the upper limit. C2 and C3 ensure that the sum of computation capacities allocated to MTDs does not exceed the total computation capacity of the edge server at the $n^{\text{th}}$ IBS. C4 and C5 ensure that each MTD chooses only one computing decision to process data. Obviously, these constraints are strongly coupled. Besides, as $o_m^j$ and $p_m$ are known only to the $m^{\text{th}}$ MTD, the joint optimization problem can be expressed as a partially observable MDP, which is a PSPACE-hard problem. It is not likely to be solved in polynomial time, and the classical model-driven algorithms require exponential complexity to solve PSPACE-hard problems (Cao et al., 2020). Thus, we use the DRL-based algorithm to address the above joint optimization problem by formulating an MDP.

### 4.2 MDP formulation

MTDs, as agents, interact with end–edge orchestrated IWN and approximate the system model. During the interaction, MTDs change their states and obtain distinct rewards by performing different actions. By maximizing the long-term cumulative reward, we can solve the joint optimization problem (8). Next, we use an MDP to model the interaction process, which is described by the state, action, reward, and state transfer function.

#### 4.2.1 State

At time-slot $t$, each MTD observes its own state $s_m(t)$ comprising the original computing decision, data size, required computation resource, and locations of IBSs. Therefore, $s_m(t)$ is expressed as

$$s_m(t)=\left\{o_m^j(t),d_m(t),c_m(t),l_m^{\mathcal{N}}(t)\right\}, \tag{9}$$

where $l_m^{\mathcal{N}}(t)$ denotes the distance between the $m^{\text{th}}$ MTD and all IBSs, indicating the locations of IBSs.

#### 4.2.2 Action

At time-slot $t$, each MTD performs action $a_m(t)$ comprising the executable computing decision $a_{m,\text{o}}(t)$ and executable transmission power $a_{m,\text{p}}(t)$. Therefore, $a_m(t)$ is expressed as

$$a_m(t)=\left\{a_{m,\text{o}}(t),a_{m,\text{p}}(t)\right\}. \tag{10}$$

Comprehensively, $a_{m,\text{o}}(t)\in\{0,1,\cdots,N\}$, where $a_{m,\text{o}}(t)=0$ indicates that the $m^{\text{th}}$ MTD processes data in end computing, and $a_{m,\text{o}}(t)=n$ indicates that the $m^{\text{th}}$ MTD processes data in edge computing (i.e., it offloads data to the edge server at the $n^{\text{th}}$ IBS). Similarly, $a_{m,\text{p}}(t)\in\{0,1,\cdots,P\}$, where $a_{m,\text{p}}(t)=0$ indicates that the $m^{\text{th}}$ MTD processes data in end computing, and $a_{m,\text{p}}(t)=p$ indicates that the $m^{\text{th}}$ MTD offloads data with transmission power $p$. Besides, if several MTDs

simultaneously offload data to the same edge server, the computation capacity of the edge server is assigned equally to these MTDs (Dai et al., 2020). Therefore, $a_m(t)$ is strictly constrained by C1–C5.

#### 4.2.3 Reward

At time-slot $t$, each MTD obtains a reward $r_m(t)$ by performing action $a_m(t)$ at state $s_m(t)$. $r_m(t)$ comprises the delay reward $r_{m,\mathrm{d}}(t)$ and energy consumption reward $r_{m,\mathrm{e}}(t)$, and it is expressed as

$$r_m(t) = \{r_{m,\mathrm{d}}(t), r_{m,\mathrm{e}}(t)\}. \tag{11}$$

Considering the joint optimization problem (8), we design the delay reward $r_{m,\mathrm{d}}(t)$ as the normalized delay difference between the end computing delay and edge computing delay, i.e., $r_{m,\mathrm{d}}(t) = \left(T_m^{\mathrm{l}} - T_m^n\right)/T_m^{\mathrm{l}}$. Similarly, we design the energy consumption reward $r_{m,\mathrm{e}}(t)$ as the normalized energy difference between the end computing energy consumption and edge computing energy consumption, i.e., $r_{m,\mathrm{e}}(t) = \left(E_m^{\mathrm{l}} - E_m^n\right)/E_m^{\mathrm{l}}$. Finally, the reward $r_m(t)$ is designed as

$$r_m(t) = \omega r_{m,\mathrm{d}}(t) + (1-\omega)\, r_{m,\mathrm{e}}(t). \tag{12}$$

As shown in Eq. (12), when data $D_m$ are processed in end computing, the reward $r_m(t)$ is 0. On the contrary, when data $D_m$ are processed in edge computing, the reward $r_m(t)$ is a non-zero real value. If the delay in edge computing is lower than that in end computing, $r_{m,\mathrm{d}}(t)$ is a positive reward; otherwise, $r_{m,\mathrm{d}}(t)$ is a negative reward. Similarly, if the energy consumption in edge computing is lower than that in end computing, $r_{m,\mathrm{e}}(t)$ is a positive reward; otherwise, $r_{m,\mathrm{e}}(t)$ is a negative reward.

Obviously, the joint optimization problem (8) is transformed into a reward with respect to delay and energy consumption. Next, by maximizing the long-term cumulative reward $R_m(t)$, we can determine an effective resource allocation policy that minimizes the system overhead, i.e.,

$$\max R_m(t) = \max \sum_{\tau=0}^{t} \gamma^{\tau} r_m(\tau), \tag{13}$$

where $\gamma$ denotes a discounted factor indicating the degree of influence of the past rewards on the current reward, $\tau$ the past time-slots, and $r(\tau)$ the reward from $s(\tau)$ to $s(\tau+1)$.

#### 4.2.4 State transfer function

At time-slot $t$, the state transfer function of the $m^{\mathrm{th}}$ MTD, namely $f_m(t)$, is the probability that the $m^{\mathrm{th}}$ MTD moves to $s_m(t+1)$ by performing $a_m(t)$ at $s_m(t)$, i.e., $f_m\left(s_m(t+1)|s_m(t), a_m(t)\right)$. With the increase in interaction, $f_m(t)$ gradually converges to an optimal state transfer function (i.e., $f_m^*(t)$) by maximizing the long-term cumulative reward:

$$f_m^*(t) \leftarrow \max \sum_{\tau=0}^{t} R_m(\tau). \tag{14}$$

## 5 Proposed MADRL-RA algorithm

With the formulated MDP, we further propose the MADRL-RA algorithm to address the joint optimization problem. MADRL-RA is an actor–critic framework based algorithm. Each MTD is an independent agent including an actor, a critic, and an experience memory. For an independent agent, the actor is used to generate action, and the critic is used to guide the actor in generating a better action. The experience memory is used to store experiences for training the actors and critics. Next, we present the detailed MADRL-RA framework and the learning process (Fig. 2).

### 5.1 Multi-agent learning process

We use deep neural networks (DNNs) as the fundamental network framework for actors, comprising estimation actor networks and target actor networks. Similarly, critics use a DNN framework comprising estimation critic networks and target critic networks. All estimation networks are policy-based DNNs, i.e., $\pi\left(s|\theta_\pi\right)$. All target networks are value-based DNNs, i.e., $Q\left(s, a|\theta_Q\right)$. Target networks are used to generate the target values for training the policies of estimation networks. For any actor or critic, the network frameworks of the estimation network and the target network are the same, but their parameters are different. Specifically, for any actor or critic, the parameter of the target network copies the historical parameter of the estimation network, namely soft update, which can effectively avoid oscillation.

#### 5.1.1 Actor network

As shown in Fig. 3, for each MTD, the input of the $m^{\mathrm{th}}$ estimation actor network is its current state

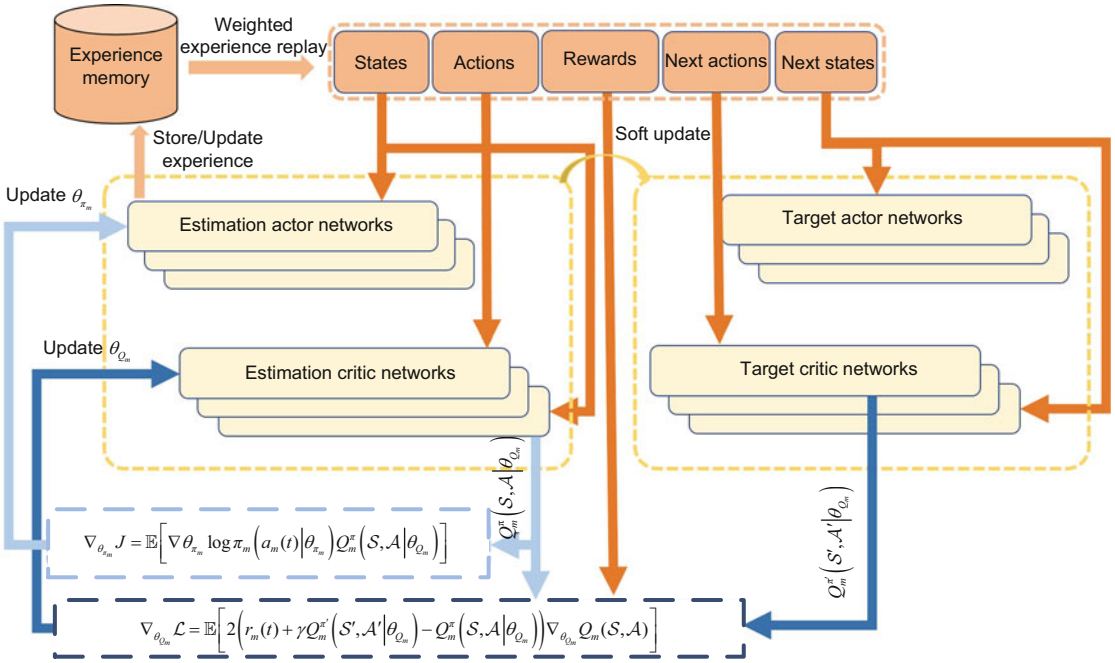


**Fig. 2 MADRL-RA framework**

$s_m(t)$. Through three fully connected layers with the rectified linear unit (ReLU) activation function, the input is transformed into the current action $a_m(t)$. Similarly, the input of the $m^{\text{th}}$ target actor network is its next state $s_m(t+1)$, which is transformed into the next action $a_m(t+1)$. In summary, each actor generates its own action $a_m$ with policy $\pi_m$ parameterized by $\theta_{\pi_m}$, i.e., $a_m = \pi_m\left(s_m|\theta_{\pi_m}\right)$.

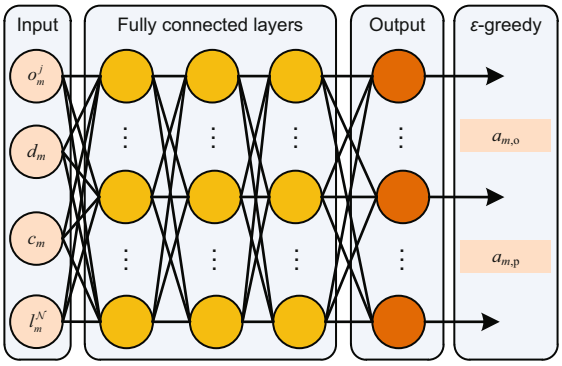


**Fig. 3 Actor network**

To learn more knowledge from the unknown IWN, MTDs need to balance exploitation and exploration. Exploitation means that MTDs use the learned knowledge by greedily taking action with the maximum value, and exploration means that MTDs acquire unknown knowledge by taking action randomly. During the learning process, we adopt the $\varepsilon$-greedy method to balance the taken action, i.e.,

$$a_m(t) = \begin{cases} a_m^{\mathrm{r}}(t), & \varepsilon, \\ a_m^{\mathrm{v}}(t), & 1-\varepsilon, \end{cases} \tag{15}$$

where $a_m^{\mathrm{r}}(t), a_m^{\mathrm{v}}(t) \in A$, and $\varepsilon$ is used to balance the ratio of exploitation and exploration. Comprehensively, $a_m^{\mathrm{r}}(t)$ denotes the exploration action, and $a_m^{\mathrm{v}}(t)$ denotes the exploitation action.

Specifically, with a small value of $\varepsilon$, MTDs prefer to perform exploitation. On the contrary, with a high value of $\varepsilon$, MTDs prefer to perform exploration. However, with a high value of $\varepsilon$ for a long time, the learning process of MADRL-RA will oscillate, making it difficult to learn an effective resource allocation policy. To explore more unknown knowledge while avoiding oscillation, we design a step-by-step $\varepsilon$-greedy method. Initially, MTDs have little knowledge, and they perform more random actions with a high value of $\varepsilon$. As MTDs gradually acquire enough knowledge, the value of $\varepsilon$ decreases, and MTDs prefer to exploit the learned knowledge and perform the action with the maximum value for learning an effective resource allocation policy. The variation of $\varepsilon$ is given by

$$\varepsilon = (1-\beta)^U \varepsilon_0, \tag{16}$$

where $0 < \varepsilon \leq 1$, $\beta$ denotes the decreased rate of exploration, $\varepsilon_0$ the initial exploration value, and $U$ the number of training times.

#### 5.1.2 Critic network

The critic network uses the $Q$-value to evaluate the performance of actions. As shown in Fig. 4, the critic network comprises an input layer, three fully connected layers with the ReLU activation function, and an output layer with one node. The input of the $m^{\text{th}}$ estimation critic network is the current states and actions of all MTDs (denoted as $\mathcal{S}$ and $\mathcal{A}$ respectively), and the output is the current $Q$-value $Q_m^{\pi}(\mathcal{S},\mathcal{A}|\theta_{Q_m})$. Similarly, the input of the $m^{\text{th}}$ target critic network is the next states and actions of all MTDs (denoted as $\mathcal{S}'$ and $\mathcal{A}'$ respectively), and the output is the next $Q$-value $Q_m^{\pi'}\left(\mathcal{S}',\mathcal{A}'|\theta'_{Q_m}\right)$. The $Q$-value is calculated and updated using the Bellman equation (Shakarami et al., 2020), expressed as

$$\begin{aligned} Q_m^{\pi}(\mathcal{S},\mathcal{A}|\theta_{Q_m}) =& Q_m^{\pi}(\mathcal{S},\mathcal{A}|\theta_{Q_m}) \\ &+ \rho(Q_{\text{tar}} - Q_m^{\pi}(\mathcal{S},\mathcal{A}|\theta_{Q_m})), \end{aligned} \tag{17}$$

where $\rho$ denotes the learning rate, $Q_{\text{tar}} = r_m(t) + \gamma \text{max} Q_m^{\pi'}\left(\mathcal{S}',\mathcal{A}'|\theta'_{Q_m}\right)$, $\mathcal{S} = \{s_1(t), s_2(t), ..., s_M(t)\}$, $\mathcal{A} = \{a_1(t), a_2(t), ..., a_M(t)\}$, $\mathcal{S}' = \{s_1(t+1), s_2(t+1), ..., s_M(t+1)\}$, and $\mathcal{A}' = \{a_1(t+1), a_2(t+1), ..., a_M(t+1)\}$.

#### 5.1.3 Parameter update

With these actions and $Q$-values, the parameters of actor networks and critic networks are updated by stochastic gradient descent. Specifically, the parameter of the $m^{\text{th}}$ estimation actor network (i.e., $\theta_{\pi_m}$) is updated as

$$\begin{aligned} &\bigtriangledown_{\theta_{\pi_m}} J \\ &\approx \mathbb{E}\left[\bigtriangledown_{\theta_{\pi_m}} \log \pi_m\left(a_m(t)|\theta_{\pi_m}\right) Q_m^{\pi}\left(\mathcal{S},\mathcal{A}|\theta_{Q_m}\right)\right]. \end{aligned} \tag{18}$$

Similarly, the parameter of the $m^{\text{th}}$ estimation critic network (i.e., $\theta_{Q_m}$) is updated by minimizing a loss function $\mathcal{L}(\theta_{Q_m})$ with respect to the current $Q$-value and the next $Q$-value. The loss function is expressed as

$$\mathcal{L}(\theta_{Q_m}) = \mathbb{E}\left[\left(Q_{\text{tar}} - Q_m^{\pi}\left(\mathcal{S},\mathcal{A}|\theta_{Q_m}\right)\right)^2\right]. \tag{19}$$

With the above loss function, $\theta_{Q_m}$ is updated by

$$\begin{aligned} &\bigtriangledown_{\theta_{Q_m}} \mathcal{L} \\ =& \mathbb{E}\left[2(Q_{\text{tar}} - Q_m^{\pi}\left(\mathcal{S},\mathcal{A}|\theta_{Q_m}\right)) \bigtriangledown_{\theta_{Q_m}} Q_m(\mathcal{S},\mathcal{A})\right]. \end{aligned} \tag{20}$$

With the increase of the number of training times, the parameters of the $m^{\text{th}}$ target actor network and the $m^{\text{th}}$ target critic network, namely $\theta'_{\pi_m}$ and $\theta'_{Q_m}$ respectively, are soft updated as

$$\theta'_{\pi_m} = \lambda\theta_{\pi_m} + (1-\lambda)\theta'_{\pi_m}, \tag{21}$$

$$\theta'_{Q_m} = \lambda\theta_{Q_m} + (1-\lambda)\theta'_{Q_m}, \tag{22}$$

where $\lambda \in [0,1]$.

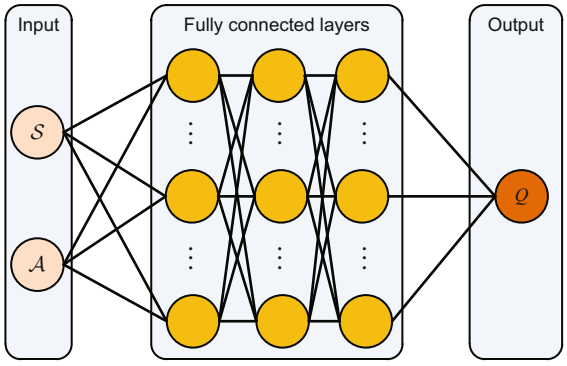


**Fig. 4 Critic network**

### 5.2 Weighted experience replay

MADRL-RA requires sufficient training data to learn an effective resource allocation policy. Thus, we use the experience $E$ comprising the state, action, reward, and next state as the training data for actors and critics. Each MTD owns an experience memory $H$ storing the experiences. At time-slot $t$, the experience of the $m^{\text{th}}$ MTD is expressed as

$$E_m(t) = \{s_m(t), a_m(t), r_m(t), s_m(t+1)\}. \tag{23}$$

Furthermore, at time-slot $t$, the experience memory $H_m(t)$ of the $m^{\text{th}}$ MTD is expressed as

$$H_m(t) = \{E_m(0), E_m(1), ..., E_m(t)\}. \tag{24}$$

To ensure the effective learning of MADRL-RA, the training data sampled from experience memory must be independent and identically distributed. Therefore, we use the experience replay to randomly sample experiences and break their time correlation.

In the classical experience replay, the sampling probability of any experience is the same. However, we consider that different experiences have different contributions toward the learning process and the convergence of MADRL-RA, and hence use the temporal difference error $\delta_h$ as the weight of the $h^{\text{th}}$ experience (Schaul et al., 2016), i.e.,

$$y_h = |\delta_h| + \zeta, \tag{25}$$

where $\zeta$ denotes a positive real value approximately equal to 0 for enabling the experience with $\delta_h = 0$ to be sampled.

To simplify the proposed experience replay, we design a weighted experience replay with two sub-experience memories (called memory $A$ and memory $B$), where memory $A$ and memory $B$ store high- and low-weight experiences, respectively. Specifically, we adopt the average weight (denoted as $\overline{y}$) to clarify the high- or low-weight experience, expressed as

$$\overline{y} = \frac{1}{H}\sum_{h=0}^{H} y_h. \tag{26}$$

If $y_h \geq \overline{y}$, the $h^{\text{th}}$ experience is stored in memory $A$; otherwise, it is stored in memory $B$. Specifically, at the beginning of training, memory $A$ and memory $B$ have the same sampling probability. To accelerate the learning process and the convergence of MADRL-RA, the sampling probability of memory $A$ is increased with the increase of the number of training times, but the sampling probability of memory $B$ decreases. Thus, the sampling probability $g_x$ is expressed as

$$g_x = g_0(1 - g_x^{\text{d}})^U, \tag{27}$$

where $x \in \{A, B\}$, $0 \leq g_x \leq 1$, $g_0$ denotes the initial sampling probability of memory $A$ or $B$, and $g_x^{\text{d}}$ denotes the sampling decay rate of memory $A$ or $B$.

### 5.3 MADRL-RA training

With the proposed weighted experience replay, experiences are sampled as training data. Among these training data, $s_m(t)$ is fed into the $m^{\text{th}}$ estimation actor network and transformed into the current action $a_m(t)$. Further, $\mathcal{S}$ and $\mathcal{A}$ are fed into the $m^{\text{th}}$ estimation critic network and transformed into the current $Q$-value $Q_m^{\pi}(\mathcal{S}, \mathcal{A}|\theta_{Q_m})$. Similarly, $s_m(t+1)$ is fed into the $m^{\text{th}}$ target actor network and transformed into the next action $a_m(t+1)$. $\mathcal{S}'$ and $\mathcal{A}'$ are fed into the $m^{\text{th}}$ target critic network and transformed into the next $Q$-value $Q_m^{\pi'}\left(\mathcal{S}', \mathcal{A}'|\theta'_{Q_m}\right)$. According to Eqs. (18)–(22), the parameters of the actors and critics are updated. Meanwhile, the experiences are updated and stored in experience memories. The complete training process is shown in Algorithm 1.

**Algorithm 1** Training process of MADRL-RA

**for** $u = 0, 1, ..., U$ **do**
  Sample $L$ experiences from memory $A$ and memory $B$ as training data;
  **for** $m = 0, 1, ..., M$ **do**
    Input $s_m(t)$ to the $m^{\text{th}}$ estimation actor network, and obtain $a_m(t) = \pi_m(s_m(t)|\theta_{\pi_m})$;
    Transfer from $s_m(t)$ into $s_m(t+1)$ by performing $a_m(t)$, and obtain reward $r_m(t)$;
    Store or update $s_m(t)$, $a_m(t)$, $r_m(t)$, and $s_m(t+1)$ in experience memories;
    Input $\mathcal{S}$ and $\mathcal{A}$ to the $m^{\text{th}}$ estimation critic network, and obtain $Q_m^{\pi}(\mathcal{S}, \mathcal{A}|\theta_{Q_m})$;
    Input $s_m(t+1)$ to the $m^{\text{th}}$ target actor network, and obtain $a_m(t+1) = \pi_m'\left(s_m(t+1)|\theta_{\pi_m'}\right)$;
    Input $\mathcal{S}'$ and $\mathcal{A}'$ to the $m^{\text{th}}$ target critic network, and obtain $Q_m^{\pi'}\left(\mathcal{S}', \mathcal{A}'|\theta'_{Q_m}\right)$;
    Update $\theta_{\pi_m}$ and $\theta_{Q_m}$;
    Set $s_m(t)$ as $s_m(t+1)$;
  **end for**
  Update $\theta'_{\pi_m}$ and $\theta'_{Q_m}$;
**end for**

Specifically, during the training of MADRL-RA, each actor needs its own state and the $Q$-value from the corresponding critic, while the critic needs the states and actions of all actors. After the training process is completed, the execution process needs only these actors, and each actor can make an effective action according to its own state.

### 5.4 Computational complexity analysis

To characterize the efficiency of MADRL-RA, we further evaluate the computational complexity. Actors and critics are all DNNs-based, and the computational complexity (Naparstek and Cohen, 2019) is expressed as

$$O(G) = O\left(\sum_{f=1}^{F} d_f d_{f+1}\right), \tag{28}$$

where $O(G)$ denotes the computational complexity of DNNs, $F$ the number of layers, and $d_f$ the number

of neurons at the $f^{\text{th}}$ layer. Specifically, the computational complexity of the actor network is expressed as $O_{\text{a}}(G)$, and the computational complexity of the critic network is expressed as $O_{\text{c}}(G)$.

In offline training, with $U$ episodes, $L$ experiences, and $M$ agents, the computational complexities of actors and critics are $O_{\text{a}}(GLU^M)$ and $O_{\text{c}}(GLU^M)$ respectively. The high computation overhead of the training process is consumed offline, which does not interfere with the real-time execution in manufacturing processes. After the offline training, MADRL-RA can be implemented for online applications with only actors. During online execution, the computational complexity of MADRL-RA is $O_{\text{a}}(G)$.

## 6 Experimental results and analysis

In this section, many experiments are performed to verify the performance of MADRL-RA with diverse experimental scenarios, run with TensorFlow-GPU-1.14.0 and Python-3.7 on a desktop powered by Intel Xeon W2245 and NVIDIA Titan RTX.

### 6.1 Experiment setup

We consider a scalable IWN with a mix of heterogeneous industrial data, and the key parameters and their values (Wei et al., 2019) are summarized in Table 1. Similarly, the parameters and their values of MADRL-RA (Cao et al., 2020) are summarized in Table 2.

To verify the effectiveness of MADRL-RA, we compare MADRL-RA with the following benchmark algorithms (Dai et al., 2020; Lu et al., 2020):

1. Full end computing (FEC): an algorithm in which all MTDs process data in end computing.

2. Nearest edge computing (NEC): an algorithm wherein all MTDs offload their data to the nearest IBSs.

3. DQN: a single-agent DRL algorithm wherein a central agent allocates resources to MTDs.

4. Greedy: using violent iteration to obtain all possible resource allocation actions and to find the minimum system overhead within these actions.

**Table 1 Key parameters of IWN**

| Parameter | Value |
|---|---|
| Number of IBSs ($N$) | 3 |
| Number of MTDs ($M$) | 10–55 |
| Path loss exponent ($\alpha$) | 3 |
| Coverage radius of IBSs ($\mu_n$) | 1 km |
| Bandwidth ($W_n$) | 10 MHz |
| Computation capacity of edge servers ($F_n$) | 1–15 GHz/s |
| Energy consumption per unit of computation capacity of edge servers ($e_n$) | 1 W/GHz |
| Local computation capacity of MTDs ($f_m^{\text{l}}$) | 0.5 GHz/s |
| Data size of MTDs ($d_m$) | $U$[500, $10^5$] KB |
| Required computation resource of MTDs ($c_m$) | $U$[1, 9] GHz |
| Noise power ($\sigma^2$) | $10^{-17}$ W |
| Maximum transmission power ($P$) | 300 mW |

**Table 2 Key parameters of MADRL-RA**

| Parameter | Value |
|---|---|
| Learning rate ($\rho$) | Actor: $10^{-4}$<br>Critic: $10^{-3}$ |
| Discount factor ($\gamma$) | 0.9 |
| Initial exploration value ($\varepsilon_0$) | 0.9 |
| Decrease rate of exploration ($\beta$) | $10^{-4}$ |
| Initial sampling probability ($g_0$) | 0.5 |
| Sampling decay rate of memory $A$ ($g_A^{\text{d}}$) | $-0.0001$ |
| Sampling decay rate of memory $B$ ($g_B^{\text{d}}$) | 0.0001 |

### 6.2 Reward analysis

Reward is the metric used to measure the effectiveness of DRL-based algorithms. Fig. 5 shows the convergence trend of MADRL-RA and DQN. With the help of weighted experience replay, MADRL-RA converges faster and obtains a higher reward than DQN. Specifically, the convergence rate of MADRL-RA is 35% higher than that of DQN, and the reward of MADRL-RA is 20% higher than that of DQN. Because FEC, NEC, and Greedy are not DRL-based algorithms, there is no reward for them in Fig. 5. Next, we adopt MADRL-RA and DQN as the representative algorithms in the DRL field to compare them with the classical FEC, NEC, and Greedy algorithms.

### 6.3 System overhead analysis

Fig. 6 describes the relationship between the system overhead and the computation capacity of edge servers, where $M = 10$ and $\omega = 0.5$. Because end computing depends only on the local computation capacity, the system overhead of FEC is a fixed value, regardless of the computation capacity of edge servers. On the contrary, with the computation capacity of edge servers increasing, the system overheads of NEC, Greedy, DQN, and MADRL-RA all gradually decrease, while the system overheads

of Greedy and MADRL-RA are always the smallest. Specifically, when the computation capacity of edge servers is weak (e.g., the computation capacity of edge servers is smaller than 2 GHz/s), the transmission cost outweighs the offloading benefit, and MTDs prefer to perform end computing, resulting in the system overhead of FEC being smaller than that of NEC. In this case, MADRL-RA can learn an effective resource allocation policy, achieving the same minimum system overhead as Greedy. However, when the computation capacity of edge servers is strong (e.g., the computation capacity of edge servers is larger than 8 GHz/s), the offloading benefit outweighs the transmission cost, and the system overhead of NEC is smaller than that of FEC. In this case, offloading data in NEC depends only on the distance between MTDs and IBSs, which triggers edge server overload and high system overhead. Fortunately, MADRL-RA can compromise between the distance and the computation capacity and learn an effective resource allocation policy, achieving smaller system overhead than NEC.

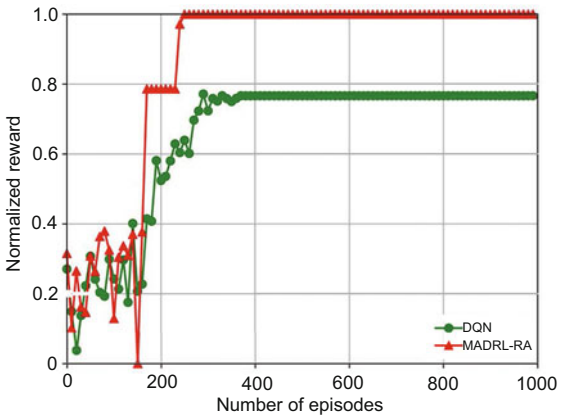


**Fig. 5 Normalized reward**

Fig. 7 shows the relationship between the system overhead and the data size, where $M = 10$, $\omega = 0.5$, and $F_n = 10$ GHz/s. Because end computing does not require transmitting data, the changing data size does not affect the system overhead of FEC. On the contrary, with the increase of data size, the system overheads of NEC, Greedy, DQN, and MADRL-RA all gradually increase, while the system overheads of MADRL-RA and Greedy are the smallest. When the data size changes from 500 to 3000 KB, the system overheads of MADRL-RA, Greedy, and DQN are much smaller than those of NEC and FEC.

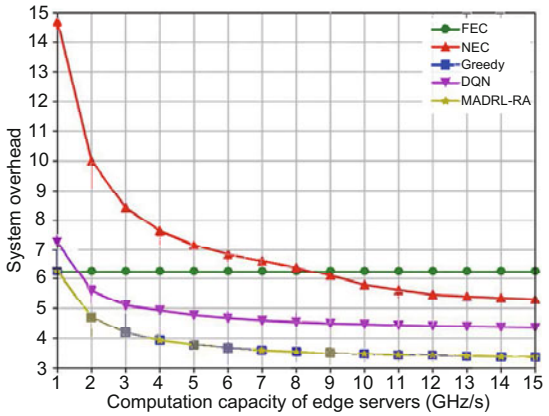


**Fig. 6 System overhead vs. computation capacity of edge servers ($M = 10$, $\omega = 0.5$)**

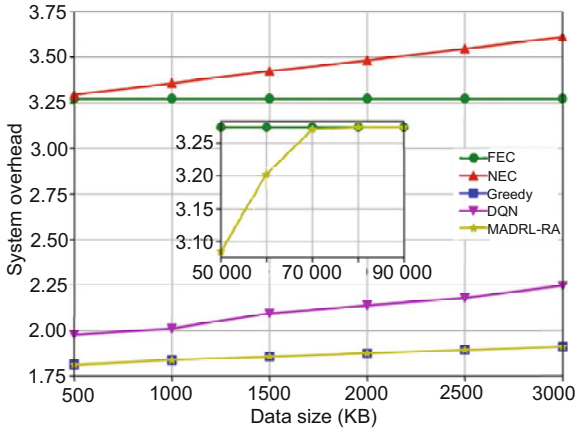


**Fig. 7 System overhead vs. data size ($M = 10$, $\omega = 0.5$, $F_n = 10$ GHz/s)**

However, when the data size is larger than 70 MB, the offloading benefit does not compensate for the transmission cost, and the system overhead of MADRL-RA approaches that of FEC.

Fig. 8 shows the relationship between the system overhead and the required computation resources, where $M = 10$, $\omega = 0.5$, and $F_n = 10$ GHz/s. With the increase of the required computation resources, the system overheads of FEC, NEC, Greedy, DQN, and MADRL-RA gradually increase, and the system overhead of MADRL-RA is always the smallest, same as that of Greedy. Specifically, when the required computation resource is smaller than 5000 MHz, the local computation capacity is sufficient for processing data in end computing. Therefore, the system overhead of FEC is smaller than that of NEC. On the contrary, when the required computation resource is larger than 5000 MHz, the strong

computation capacity of edge servers provides high offloading benefit, which makes the system overhead of NEC become smaller than that of FEC.

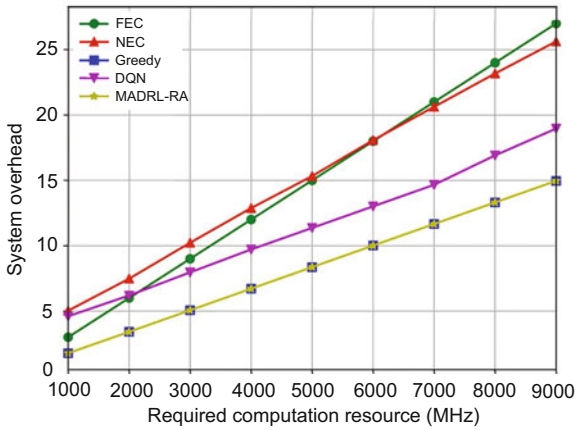


**Fig. 8 System overhead vs. required computation resources ($M = 10$, $\omega = 0.5$, $F_n = 10$ GHz/s)**

Fig. 9 describes the relationship between the system overhead and the number of MTDs, where $F_n = 10$ GHz/s. With the increase in the number of MTDs, the system overheads of FEC, NEC, Greedy, DQN, and MADRL-RA gradually increase. Specifically, when the number of MTDs is around 10, the computation capacity that MTDs obtain from edge servers is roughly equal to their local computation capacity. Therefore, in this case, the system overheads of FEC, NEC, Greedy, DQN, and MADRL-RA are similar. However, when the number of MTDs is larger than 15, the high concurrent offloading of massive MTDs may result in edge server overload. In this case, the computation capacity that MTDs obtain from edge servers is smaller than their local computation capacity. Thus, the system overhead of NEC increases significantly. When the number of MTDs exceeds 30, the offloading benefit does not compensate for the transmission cost, and the system overheads of FEC, DQN, and MADRL-RA are the same; i.e., the learned effective resource allocation policy of MADRL-RA is FEC.

### 6.4 Weight factor analysis

To verify the effect of $\omega$ on the computing decision and transmission power of MTDs with different data sizes and required computation resources, we consider four types of data (Table 3).

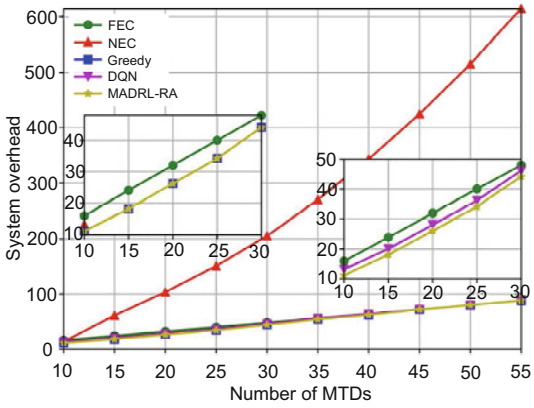


**Fig. 9 System overhead vs. the number of MTDs ($F_n = 10$ GHz/s)**

**Table 3 Data types**

| Type | $d_m$ (MB) | $c_m$ (MHz) |
|---|---|---|
| Type 1 | 100 | 5000 |
| Type 2 | 100 | 500 |
| Type 3 | 10 | 5000 |
| Type 4 | 10 | 500 |

Fig. 10 describes the relationships between the end computing ratio and $\omega$ and between the edge computing ratio and $\omega$, where the solid and dashed lines represent the end computing ratio and edge computing ratio, respectively. Comprehensively, the end computing ratio represents the ratio of the number of MTDs in end computing to the total number of MTDs, and the edge computing ratio represents the ratio of the number of MTDs in edge computing to the total number of MTDs. As $\omega$ increases, delay becomes increasingly dominant in the system overhead. Type 1 and type 3 all require high computation resources, but they differ in data size. Specifically, with $\omega$ increasing, type 1 and type 3 prefer edge computing to obtain strong computation capacity, reducing the processing delay; i.e., edge computing ratios of type 1 and type 3 gradually increase. Because the data size of type 3 is smaller than that of type 1, the edge computing ratio of type 3 is higher than that of type 1. On the contrary, type 2 and type 4 all require low computation resources and differ in data size. With $\omega$ increasing, because the data size of type 2 is higher than that of type 4, type 2 prefers end computing to reduce the transmission delay. Thus, the end computing ratio of type 2 gradually increases and is higher than that of type 4.

Correspondingly, Fig. 11 describes the relationship between the transmission power ratio and the

value of $\omega$, where the transmission power ratio represents the ratio of the transmission power used by MTDs to the maximum transmission power (i.e., $p_{\mathrm{m}}/P$). We still consider the four types of data in Table 3. Type 1 and type 3 require high computation resources and prefer edge computing, resulting in high transmission power ratios. As $\omega$ increases, the transmission power ratios of type 1 and type 3 also increase. Specifically, because the data size of type 3 is smaller than that of type 1, the edge computing ratio of type 3 is higher, resulting in the higher transmission power ratio of type 3. Type 2 and type 4 require low computation resources and prefer end computing, resulting in smaller transmission power ratios compared with type 1 and type 3. Specifically, because the data size of type 2 is larger than that of type 4, the end computing ratio of type 2 is higher, resulting in the smaller transmission power ratio of type 2.

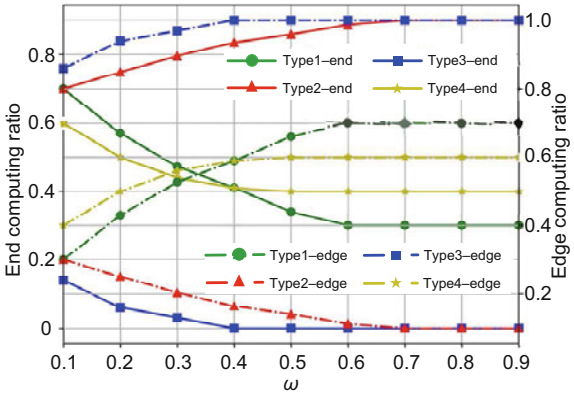


**Fig. 10 End computing ratio and edge computing ratio vs. $\omega$**

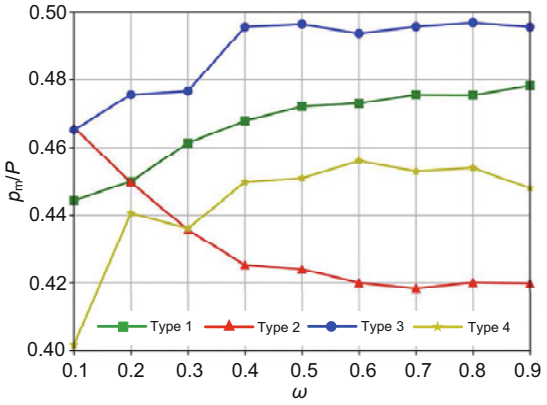


**Fig. 11 Transmission power ratio vs. $\omega$**

## 7 Conclusions

In this study, we regarded distributed MTDs as multiple self-learning agents to deal with dynamic and time-varying end–edge orchestrated IWNs, and proposed the MADRL-RA algorithm to learn an end–edge orchestrated resource allocation policy to minimize system overhead with respect to delay and energy consumption. Compared with FEC, NEC, and DQN, MADRL-RA can learn the effective resource allocation policy and adapt to changes in the computation capacity of edge servers, data size, required computation resource, and number of MTDs. Moreover, by setting different weight factors, we can optimize resource allocation to satisfy heterogeneous industrial applications in terms of different delay or energy consumption levels. Compared with model-driven algorithms, the offline training of MADRL-RA requires a certain computation overhead, but the online execution of MADRL-RA can achieve flexible and autonomous resource allocation with low computational complexity. In future work, we will apply MADRL-RA in practical IWNs for real end–edge orchestrated resource allocation.

### Contributors

Xiaoyu LIU, Chi XU, and Haibin YU designed the research. Xiaoyu LIU processed the data and drafted the paper. Chi XU, Haibin YU, and Peng ZENG helped organize the paper. Xiaoyu LIU and Chi XU revised and finalized the paper.

### Compliance with ethics guidelines

Xiaoyu LIU, Chi XU, Haibin YU, and Peng ZENG declare that they have no conflict of interest.

### References

Alfakih T, Hassan MM, Gumaei A, et al., 2020. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. *IEEE Access*, 8:54074-54084. https://doi.org/10.1109/access.2020.2981434

Cao ZL, Zhou P, Li RX, et al., 2020. Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in Industry 4.0. *IEEE Int Things J*, 7(7):6201-6213. https://doi.org/10.1109/jiot.2020.2968951

Chen Y, Liu ZY, Zhang YC, et al., 2021. Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial Internet of Things. *IEEE Trans Ind Inform*, 17(7):4925-4934. https://doi.org/10.1109/tii.2020.3028963

Chu TS, Wang J, Codecà L, et al., 2020. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Trans Intell Transp Syst*, 21(3):1086-1095. https://doi.org/10.1109/tits.2019.2901791

Dai YY, Zhang K, Maharjan S, et al., 2020. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. *IEEE Trans Veh Technol*, 69(10):12175-12186. https://doi.org/10.1109/tvt.2020.3013990

Feng J, Pei QQ, Yu FR, et al., 2019. Computation offloading and resource allocation for wireless powered mobile edge computing with latency constraint. *IEEE Wirel Commun Lett*, 8(5):1320-1323. https://doi.org/10.1109/lwc.2019.2915618

Foerster JN, Assael YM, de Freitas N, et al., 2016. Learning to communicate with deep multi-agent reinforcement learning. Proc Advances in Neural Information Processing Systems 29, p.2137-2145.

Guo JF, Song ZZ, Cui Y, et al., 2017. Energy-efficient resource allocation for multi-user mobile edge computing. Proc IEEE Global Communications Conf, p.1-7.

He XM, Lu HD, Du M, et al., 2021. QoE-based task offloading with deep reinforcement learning in edge-enabled Internet of Vehicles. *IEEE Trans Intell Transp Syst*, 22(4):2252-2261. https://doi.org/10.1109/tits.2020.3016002

Kumar M, Sharma SC, Goel A, et al., 2019. A comprehensive survey for scheduling techniques in cloud computing. *J Netw Comput Appl*, 143:1-33.

Li HL, Xu HT, Zhou CC, et al., 2020. Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment. *IEEE Trans Veh Technol*, 69(9):10214-10226. https://doi.org/10.1109/tvt.2020.3003898

Lin CC, Deng DJ, Chih YL, et al., 2019. Smart manufacturing scheduling with edge computing using multiclass deep *Q* network. *IEEE Trans Ind Inform*, 15(7):4276-4284. https://doi.org/10.1109/tii.2019.2908210

Liu KH, Liao WJ, 2020. Intelligent offloading for multi-access edge computing: a new actor-critic approach. Proc IEEE Int Conf on Communications, p.1-6.

Liu XY, Xu C, Yu HB, et al., 2021. Deep reinforcement learning-based multi-channel access for industrial wireless networks with dynamic multi-user priority. *IEEE Trans Ind Inform*, early access. https://doi.org/10.1109/TII.2021.3139349

Lowe R, Wu Y, Tamar A, et al., 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. Proc Advances in Neural Information Processing Systems 30, p.6379-6390.

Lu HD, He XM, Du M, et al., 2020. Edge QoE: computation offloading with deep reinforcement learning for Internet of Things. *IEEE Int Things J*, 7(10):9255-9265. https://doi.org/10.1109/jiot.2020.2981557

Naparstek O, Cohen K, 2019. Deep multi-user reinforcement learning for distributed dynamic spectrum access. *IEEE Trans Wirel Commun*, 18(1):310-323. https://doi.org/10.1109/TWC.2018.2879433

Porambage P, Okwuibe J, Liyanage M, et al., 2018. Survey on multi-access edge computing for Internet of Things realization. *IEEE Commun Surv Tut*, 20(4):2961-2991. https://doi.org/10.1109/comst.2018.2849509

Rashid T, Samvelyan M, de Witt CS, et al., 2018. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. Proc 35th Int Conf on Machine Learning, p.4292-4301.

Ren YJ, Sun YH, Peng MG, 2021. Deep reinforcement learning based computation offloading in fog enabled industrial Internet of Things. *IEEE Trans Ind Inform*, 17:4978-4987. https://doi.org/10.1109/tii.2020.3021024

Schaul T, Quan J, Antonoglou I, et al., 2016. Prioritized experience replay. Proc 4th Int Conf on Learning Representations.

Shakarami A, Ghobaei-Arani M, Shahidinejad A, 2020. A survey on the computation offloading approaches in mobile edge computing: a machine learning-based perspective. *Comput Netw*, 182:107496. https://doi.org/10.1016/j.comnet.2020.107496

Tang L, He SB, 2018. Multi-user computation offloading in mobile edge computing: a behavioral perspective. *IEEE Netw*, 32(1):48-53. https://doi.org/10.1109/mnet.2018.1700119

Wang HN, Liu N, Zhang YY, et al., 2020. Deep reinforcement learning: a survey. *Front Inform Technol Electron Eng*, 21:1726-1744. https://doi.org/10.1631/FITEE.1900533

Wei YF, Yu FR, Song M, et al., 2019. Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning. *IEEE Int Things J*, 6(2):2061-2073. https://doi.org/10.1109/jiot.2018.2878435

Xiong X, Zheng K, Lei L, et al., 2020. Resource allocation based on deep reinforcement learning in IoT edge computing. *IEEE J Sel Area Commun*, 38(6):1133-1146. https://doi.org/10.1109/jsac.2020.2986615

Xu C, Zeng P, Yu HB, et al., 2021. WIA-NR: ultra-reliable low-latency communication for industrial wireless control networks over unlicensed bands. *IEEE Netw*, 35(1):258-265. https://doi.org/10.1109/mnet.011.2000308

Yao XF, Zhou JJ, Lin YZ, et al., 2019. Smart manufacturing based on cyber-physical systems and beyond. *J Intell Manuf*, 30(8):2805-2817. https://doi.org/10.1007/s10845-017-1384-5

Yu HB, Zeng P, Xu C, 2021. Industrial wireless control networks: from WIA to the future. *Engineering*, early access. https://doi.org/10.1016/j.eng.2021.06.024

Zhang GL, Zhang WQ, Cao Y, et al., 2018. Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices. *IEEE Trans Ind Inform*, 14(10):4642-4655. https://doi.org/10.1109/tii.2018.2843365

Zhang KQ, Yang ZR, Basar T, 2021. Decentralized multi-agent reinforcement learning with networked agents: recent advances. *Front Inform Technol Electron Eng*, 22:802-814. https://doi.org/10.1631/FITEE.1900661

Zhang P, Peng MG, Cui SG, et al., 2022. Theory and techniques for "intellicise" wireless networks. *Front Inform Technol Electron Eng*, 23(1):1-4. https://doi.org/10.1631/FITEE.2210000

Zhang YM, Lan XL, Ren J, et al., 2020. Efficient computing resource sharing for mobile edge-cloud computing networks. *IEEE/ACM Trans Netw*, 28(3):1227-1240. https://doi.org/10.1109/tnet.2020.2979807

Zhu XY, Luo YY, Liu AF, et al., 2021. Multiagent deep reinforcement learning for vehicular computation offloading in IoT. *IEEE Int Things J*, 8(12):9763-9773. https://doi.org/10.1109/jiot.2020.3040768