# QDRL: Queue-Aware Online DRL for Computation Offloading in Industrial Internet of Things

Aikun Xu, Zhigang Hu, Xinyu Zhang, Hui Xiao, Hao Zheng, *Graduate Student Member, IEEE*, Bolei Chen, Meiguang Zheng, Ping Zhong, *Member, IEEE*, Yilin Kang, and Keqin Li, *Fellow, IEEE*

***Abstract*—Recently, the Industrial Internet of Things (IIoT) has shown great application value in environmental monitoring. However, it suffers from serious bottlenecks in energy and computing capability. To address them, researchers have made lots of effort. Nevertheless, they neglect either the edge–end collaboration or the impact of task queue backlog, resulting in low system revenue. To this end, we design a queue-aware computation offloading method based on DRL (QDRL). Specifically, we represent the long-term system operation as a multistage stochastic mixed-integer optimization problem (M-SMIP), which is further converted into a deterministic problem using Lyapunov optimization. Given that the resource allocation and computation offloading in this deterministic problem are strongly coupled and difficult to solve, we decompose this problem into two subproblems. Subsequently, a reinforcement learning scheme with actor–critic architecture is designed to solve these subproblems. The Actor module is designed based on a deep learning model and quantization strategy for generating computation offloading actions. The mathematical reasoning and learning-based methods are integrated as the Critic module for achieving resource allocation. Extensive simulation results show that the performance of QDRL surpasses four baselines and approaches the approximate optimal algorithm in terms of average task queue length, normalized real computation rate, and computation time.**

***Index Terms*—Computation offloading, deep learning, edge computing, Industrial Internet of Things (IIoT), reinforcement learning (RL).**

## I. Introduction

THE ACCELERATED advancement in hardware, software, and wireless communication technologies has given birth to the Internet of Things (IoT) [1]. The combination of IoT technology and industrial device constitutes the Industrial IoT (IIoT), which is a hot topic [2], [3]. In recent years, the application scenarios of the IIoT have gradually shifted from indoor (e.g., product quality inspection) to outdoor (e.g., industrial environmental monitoring) [4]. Unlike the former, the latter usually applies IIoT devices (i.e., wireless sensors) to monitor the industrial environment. These devices are battery-powered and limited by battery life for long-term operation. Moreover, the computing capability of the device is usually low and cannot process lots of computing-intensive tasks [5]. To address the above challenges, researchers have done a lot of effort. On the one hand, the development of wireless transmission power (WTP) technology enables IIoT devices to be charged wirelessly without frequent battery replacement. The WTP has reached tens of microwatts at a distance of more than 10 (unit meter), and it will even be higher in the future, which is enough for many low-power IIoT devices to provide sufficient power [6]. On the other hand, the introduction of emerging technologies (e.g., cloud computing) has provided IIoT with sufficient resources far exceeding the IIoT device, making it possible to handle large-scale computing-intensive tasks.

Deep learning task (DLT) (e.g., industrial environmental monitoring) is a computing-intensive task that needs to be processed (i.e., inference or prediction) with a deep learning model, e.g., fully connected neural network. In IIoT, the generation and arrival of DLT is usually dynamic and stochastic [7]. Dealing with DLT usually brings benefits to society, e.g., improving air quality, improving employee safety, reducing human and material investment, etc. [1]. To obtain more benefits, preliminary solutions offload the DLT to the central cloud with sufficient resources for processing [8], [9], [10]. DLTs in IIoT have stringent requirements for real-time performance. However, due to the large distance between the central cloud and users, it becomes challenging to fulfill the requirements for low latency in data transmission and communication.

As a transformative computing paradigm, edge computing transposes the computing capability from the centralized cloud to the network's edge [11], [12]. This shift offers the advantage of enabling shorter response time, which can solve the problem of slow response in the central cloud [13]. The edge server with rich resources, which can deploy the deep learning model with more hidden layers. In contrast, IIoT devices can only deploy the shallow deep learning model due to resource constraints [14]. In general, the accuracy of deep learning models with more hidden layers is usually higher than shallow

Manuscript received 3 July 2023; revised 26 August 2023; accepted 12 September 2023. Date of publication 15 September 2023; date of current version 21 February 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62172442 and Grant 62272489, and in part by the National Science Foundation of Hunan Province under Grant 2022JJ30760. *(Corresponding author: Zhigang Hu.)*

Aikun Xu, Zhigang Hu, Xinyu Zhang, Hui Xiao, Hao Zheng, Bolei Chen, Meiguang Zheng, and Ping Zhong are with the School of Computer Science and Engineering, Central South University, Changsha 410073, China (e-mail: aikunxu@csu.edu.cn; zghu@csu.edu.cn; zhangxinyu11014@163.com; huixiao@csu.edu.cn; zhenghao@csu.edu.cn; boleichen@csu.edu.cn; zhengmeiguang@csu.edu.cn; ping.zhong@csu.edu.cn).

Yilin Kang is with the School of Computer Science, South Central University for Nationalities, Wuhan 430074, China (e-mail: ylkang@mail.scuec.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York at New Paltz, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/JIOT.2023.3316139

deep learning models. If all DLTs are offloaded to the edge server, while it can satisfy high-accuracy requirements, the system may struggle to process new DLTs generated by IIoT devices in a timely manner. This could lead to a backlog in task queues and potential system instability. If all DLTs are only processed in IIoT devices, it will be difficult to meet the high-accuracy requirements (i.e., low inference accuracy, means appear a lot of errors), thus increasing the economic loss of the system [15]. Therefore, determining which DLTs to process locally or offload to the edge server is a key issue. To address the above problem, researchers considered the static IIoT scenario and proposed a one-shot optimization solution (i.e., the parameters will not change after deployment) [16]. However, the industrial environment is dynamically changing (e.g., the wireless channel is dynamically changing). The traditional algorithms that only consider one-shot optimization will no longer be applicable [17]. Moreover, randomly generated and arriving DLTs will enter the task queue of IIoT devices to be processed. If these DLTs are handled improperly, there will be a queue backlog, causing system instability and unnecessary losses [18].

To solve the above problems, we use the access point (AP) to achieve power supply to IIoT devices through the WTP and design a queue-aware computation offloading algorithm based on DRL (QDRL). Specifically, we first model the long-term system operation as an multistage stochastic mixed-integer optimization problem (M-SMIP) problem and convert it to a deterministic problem. Subsequently, this deterministic problem is broken down into two subproblems, i.e., the computation offloading subproblem and the resource allocation subproblem, which are solved by an actor–critic (AC)-based solution. Finally, the simulation results show the advantage of QDRL.

The key contributions of our study can be outlined as follows.

1) We propose QDRL, which can not only realize online computation offloading but also interact with the environment to achieve long-term stable operation of the system.
2) We introduce the Lyapunov optimization technique to convert the M-SMIP problem in (15) into a per-frame deterministic problem and adaptively adjust the task queue in the system.
3) We design the DNN to predict the preliminary computation offloading action and present COGA to enrich the set of candidate offloading actions. Subsequently, a clever fusion of mathematical reasoning and learning-based approaches ensures efficient resource allocation, while maintaining a high real computation rate (RCR).
4) We conduct extensive simulations, and the results show that the performance of QDRL is better than four baseline algorithms and approaches the approximate optimal algorithm in terms of average task queue length and normalized RCR. Last but not least, QDRL can still maintain good performance in terms of computation time, which shows that QDRL is promising to be deployed in real scenarios.

The structure of this article is as follows. Section II provides a review of relevant literature. The system model and problem formulation are discussed in Section III. Section IV details the problem transformation and design of the algorithm. Performance evaluation of the QDRL is conducted in Section V. Finally, this article is concluded in Section VI.

## II. Related Works

Deep learning technology (e.g., DNN) is widely used in IIoT, e.g., product quality inspection [19], factory dangerous behavior detection [20], and industrial dust detection [21]. Unlike traditional tasks, DLTs are computing-intensive tasks and require processed (i.e., inference or prediction) with deep learning models [22], [23]. Processing DLTs through deep learning models considering different hidden layers usually obtains results with different accuracy, which is difficult to achieve for the traditional solutions. In recent, researchers have proposed some effective solutions to process DLTs, e.g., the model compression techniques and DNN partitioning techniques are applied to speed up the processing of DLTs [24]. The model compression is an efficient solution to enable deep learning models in resource-constrained devices. The DNN partitioning technology strategically segments the DNN into multiple parts in accordance with its multilayered structure, thereby substantially diminishing the task delay. Nevertheless, modifying the DNN model structure in the above ways will reduce the ability of model inference, e.g., the decrease of accuracy.

Different from the above methods, the computation offloading method based on edge computing provides us with a new way to process DLTs in IIoT, which is efficient and fast [25]. The optimization objective in computation offloading is generally to minimize task processing delay, reducing system energy consumption, or maximize computation rate (CR) [9], [10], [26]. Tang et al. [10] considered various factors (i.e., task dependence, data transmission, response time, and cost) and formulated the computation offloading problem as an energy optimization problem. The above problem was solved by the genetic algorithm. Fantacci and Picano [9] aimed to minimize the average system response time in IIoT with federated learning. Bi and Zhang [26] proposed an approximation algorithm, namely, CD, which aims to find a local optimum of the CR by iteratively flipping the user's binary offloading actions. Regrettably, the inference accuracy for DLTs has not been considered in the above works, and thus these works cannot handle tasks with specific accuracy requirements.

Inference accuracy is an important metric in IIoT, which represents the rationality of inference results as well as affects service quality and user experience [14], [27]. Yang et al. [14] deployed a shallow DNN model and a DNN model with a large number of hidden layers on the device and the edge as well as proposed a DLT scheduling scheme to optimize delay. For the scenario of multitask and multitype DNN, a scheme was proposed to realize collaborative reasoning of DLTs and allocation of computing resources in [27] considering adaptive sampling rate. However, their need for sufficient energy makes them difficult to deploy outdoors, where the outdoor IIoT is more energy-constrained.

The IIoT needs to meet some important requirements after deployment, e.g., making intelligent decisions in real


Fig. 1. Example scenario of the IIoT-based industrial environment monitoring system, where $N = 9$. QDRL does not offload all data from the sensor to the edge server (e.g., $s_1$), but cleverly leaves some data locally for processing (e.g., $s_5$) to achieve edge–end collaboration, thereby meeting the requirement for rapid response.

time, completing parameter updates autonomously, adapting to dynamically changing environments, etc. [2]. Those requirements gave rise to reinforcement learning (RL) or deep RL (DRL), e.g., $Q$-learning [28], deep-$Q$-network (DQN) [29], AC [5], and deep deterministic policy gradient (DDPG) [30]. In recent, RL has been widely used in IIoT, e.g., content caching, resource allocation, computation offloading, etc. Specifically, a deep weighted $Q$-learning algorithm is proposed to learn dynamic caching strategies to achieve the goal of minimizing latency [28]. Liu et al. [29] designed a mobile edge computing network for maximizing long-term computing resource utilization based on DQN. Huang et al. [5] proposed DROO based on AC for achieving computation offloading. DROO used a DNN in the Actor module to produce a set of 0/1 offloading decisions. The best offloading action was determined in the Critic module by addressing the problem of optimal resource allocation. The DDPG algorithm was applied to solve the joint resource allocation problem in the Internet of vehicles [30]. However, the above schemes cannot meet the constraints of long-term stability of the system, which decrease the system revenue [27].

## III. System Model and Problem Formulation

### A. System Model

Fig. 1 illustrates an IIoT-based industrial environment monitoring system, which consists of one edge server, one AP, and $N$ wireless sensors (i.e., wireless devices). Here, the set of wireless sensors is denoted as $S = \{s_1, s_2, \ldots, s_N\}$, which can be used to monitor the environment, collect data, and process data. Each sensor is equipped with a task queue and the data (i.e., task) arrival rate of sensor $s_i$ is $DA_i$. When the data arrival rate is too high, there may be a queue backlog. The AP (bound with the edge server) and the sensor communicate with each other. The AP not only supplies energy for the sensor but also receives and processes the data from the sensor and returns the result to the sensor. Within each frame duration of $T$, the gain of the wireless channel between the AP and sensor $s_i$ is denoted by $g_i$, which is assumed to be static in per-frame [5]. $g_i$ may be different in different time frames.

Every sensor deploys a shallow deep learning model (i.e., DNN1) with an inference accuracy of $A_1$. When the energy of the sensor is sufficient, the DNN1 will infer the instruction from the data to control related equipments. The edge server deploys a full deep learning model (i.e., DNN2) for inference on sensor data with an inference accuracy of $A_2$, where $A_2 > A_1$ and both DNN1 and DNN2 are trained for inference only. In this article, each sensor completes communication and energy harvesting in a time-division multiplexing (TDM) manner to avoid channel interference. The AP also adopts a similar TDM manner to achieve energy transmission and communication with the sensor.

Although the sensor can respond faster than the edge server, it cannot process all tasks due to its limited energy. Moreover, limited by inference accuracy, sensors are usually unable to handle tasks with high accuracy requirements locally. Therefore, some tasks need to be offloaded to the edge server for processing. Nevertheless, a large number of tasks transmitted to the edge server from sensors will increase the delay, which reduces system revenue, e.g., the CR (i.e., the amount of data processed per second, the unit is bit per second [31]). Therefore, we need to determine which tasks need to be processed locally (e.g., $s_5$) or offloaded to the edge server (e.g., $s_1$) to increase system revenue.

### B. Computing Mode

The IIoT for monitoring the industrial environment is usually implemented by wireless sensor networks deployed outdoors and requires long-term operation. However, these sensors are limited by battery capacity and cannot achieve a long-term operation. To address the challenge, we use AP to replenish energy for all sensors by WTP technology. The sensor $s_i$ harvests energy at the $t$th time frame is modeled as

$$E^t_{\text{har}_i} = \mu P g^t_i \alpha^t T \tag{1}$$

where $\mu$ represents the efficiency of energy harvesting, which lies between 0 and 1 [32]. $P$ denotes the transmit power of the AP's radio frequency energy. $\alpha^t T$ denotes the time of broadcasting energy from AP to all sensors at the $t$th time frame. $\alpha$ is the percentage coefficient of the AP broadcasting energy time with respect to duration $T$.

*1) Local Computing:* In this study, we adopt the model from [31] for the power consumption of a sensor's processor, which is given as $k_i f_i^3$ (the unit is joule per second). Here, $k_i$ signifies the coefficient of computation energy efficiency, and $f_i$ refers to the processor chip's computing frequency. The energy consumption of sensor $s_i$ during the $t$th time frame is represented as

$$E^t_{\text{con}_i} = k^t_i \left(f^3_i\right)^t \tau^t_{L,i} T \tag{2}$$

where $\tau_{L,i}^t T$ is the operation time of the sensor $s_i$ at the $t$th time frame. $\tau_{L,i}$ is the percentage coefficient of the sensor $s_i$ operation time with respect to duration $T$.

Similar to [5], to ensure the sensor $s_i$ normal operation, we also assume that all the harvesting energy of the sensor $s_i$ at the $t$th time frame is used to process the data, then we have

$$E_{\text{har}_i}^t = E_{\text{con}_i}^t. \tag{3}$$

In fact, the sensor $s_i$ consumes energy all the time in the duration $T$, i.e., it meets $\tau_{L,i} = 1$. Therefore, the sensor $s_i$ consumes energy at the $t$th time frame is $E_{\text{con}_i}^t = k_i^t (f_i^3)^t T$ in (2), which combines (1) and (3), we have

$$f_i^t = \left(\frac{\mu P g_i^t \alpha^t}{k_i}\right)^{\frac{1}{3}}. \tag{4}$$

The amount of data (in bit) processed in the sensor $s_i$ is modeled as

$$\text{DP}_{L,i}^t = \frac{f_i^t T}{\phi}. \tag{5}$$

Substituting (4) into (5), we have

$$\text{DP}_{L,i}^t = \frac{\left(\mu P g_i^t \alpha^t / k_i\right)^{\frac{1}{3}} T}{\phi}. \tag{6}$$

*2) Edge Computing:* Based on the Shannon capacity formula, the data (in a bit) that needs to be transmitted from the sensor $s_i$ (i.e., the device) to the edge server (i.e., the sensor $s_i$ needs offloaded data amount) is modeled as

$$\text{DP}_{O,i}^t = \frac{B\tau_{O,i}^t T}{v_u} \log_2\left(1 + \frac{P_i^t g_i^t}{N_0}\right) \tag{7}$$

where $B$ is the transmission bandwidth between any one sensor and the AP. $v_u$ is the communication overhead, where $v_u \geq 1$. $\tau_{O,i}^t T$ and $P_i^t$ represent the transmit time and transmit power of the sensor $s_i$ at the $t$th time frame, respectively. $\tau_{O,i}$ is the percentage coefficient of the sensor $s_i$ transmit time with respect to duration $T$. $N_0$ represents the noise power (in watt). Moreover, $\sum_i^N \tau_{O,i}^t + \alpha^t \leq 1$ holds when the sensor $s_i$ needs to offload its data to the edge server.

Same assumption as in [33], we also assume that the sensor $s_i$ exhausts its harvested energy when offloading, i.e., $P_i^{*t} = E_{\text{har}_i}^t / (\tau_{O,i}^t T)$, where $P_i^{*t}$ represents the maximize transmit power of the sensor $s_i$ at the $t$th time frame. Substituting $P_i^{*t}$ into (7), we have

$$\text{DP}_{O,i}^t = \frac{B\tau_{O,i}^t T}{v_u} \log_2\left(1 + \frac{\mu P \alpha^t \left(g_i^t\right)^2}{\tau_{O,i}^t N_0}\right). \tag{8}$$

In this study, we consider 0/1 offloading mode, i.e., at the $t$th time frame, the sensor $s_i$ can either locally process the collected data (i.e., task) or offload it to the edge server. Therefore, the data (in a bit) to be processed at the $t$th time frame can be expressed as

$$\text{DP}_i^t = \left(1 - x_i^t\right)\text{DP}_{L,i}^t + x_i^t \text{DP}_{O,i}^t \tag{9}$$

where $x_i^t = 1$ represents the data of sensor $s_i$ is offloaded to the edge server at the $t$th time frame. The data of sensor $s_i$ is processed in local when $x_i^t = 0$, i.e., the task will be processed in the sensor $s_i$.

The CR represents a direct metric of the system's computing capability, whose unit is bit per second [26]. The CR at the $t$th time frame can be expressed as

$$\text{CR}_i^t = \frac{\text{DP}_i^t}{T}. \tag{10}$$

Substituting (7)–(9) into (10), we have

$$\text{CR}_i^t = \left(1 - x_i^t\right)\frac{\left(\mu P g_i^t \alpha^t / k_i\right)^{\frac{1}{3}}}{\phi} + x_i^t \frac{B\tau_{O,i}^t}{v_u} \log_2\left(1 + \frac{\mu P \alpha^t \left(g_i^t\right)^2}{\tau_{O,i}^t N_0}\right). \tag{11}$$

As far as we know, there will be errors in DNN model inference, which will cause economic losses to the system. (The CR loss is used to represent the economic loss in this article.) The errors of DNN1 and DNN2 are denoted as $1-A_1$ and $1-A_2$, respectively. Therefore, the CR losses caused by processing the data collected by the sensor $s_i$ at the $t$th time frame is denoted as

$$\text{ERR}_i^t = \left(\left(1 - x_i^t\right)\left(1 - A_1\right) + x_i^t\left(1 - A_2\right)\right)\text{DA}_i^t \gamma \tag{12}$$

where $\gamma$ is a penalty factor.

This article aims to improve the CR while ensuring low CR losses. To this end, we define the RCR (the unit is bit per second), which is expressed as

$$\text{RCR}_i^t = \text{CR}_i^t - \text{ERR}_i^t. \tag{13}$$

In this article, we define the task queue length of the sensor $s_i$ as $Z_i(t)$ at the $t$th time frame, which plays a crucial role in influencing the overall system's stability [18]. The variations in the task queue can be represented as

$$Z_i(t+1) = Z_i(t) - \text{DP}_i^t + \text{DA}_i^t \tag{14}$$

where $Z_i(1) = 0$, indicating that there is no backlog in the current task queue.

### C. Problem Formulation

This study aims to maximize the total RCR through reasonable scheduling while ensuring the system stability and low CR losses. We model the optimization problem as

$$\underset{\mathbf{x},\alpha,\boldsymbol{\tau}}{\text{maximize}} \quad \lim_{K\to\infty} \frac{1}{K} \cdot \sum_{t=1}^{K}\sum_{i=1}^{N} w_i \text{RCR}_i^t \tag{15}$$

$$\text{subject to} \quad CR_i^t \leq Z_i(t) \quad \forall i \in N \tag{15a}$$

$$\lim_{K\to\infty} \frac{1}{K} \cdot \sum_{t=1}^{K} \mathbb{E}[Z_i(t)] < \infty \quad \forall i \in N \tag{15b}$$

$$\sum_{i=1}^{N} \tau_{O,i}^t + \alpha^t \leq 1 \tag{15c}$$

$$\alpha^t \geq 0, \tau_{O,i}^t \geq 0 \quad \forall i \in N \tag{15d}$$

$$x_i^t \in \{0,1\} \quad \forall i \in N \tag{15e}$$

where $w_i > 0$ is the weight of the sensor $s_i$. Equation (15a) implies that the volume of data processed in the current time frame must not surpass the length of the task queue. Equation (15b) sets a constraint on the long-term stability of

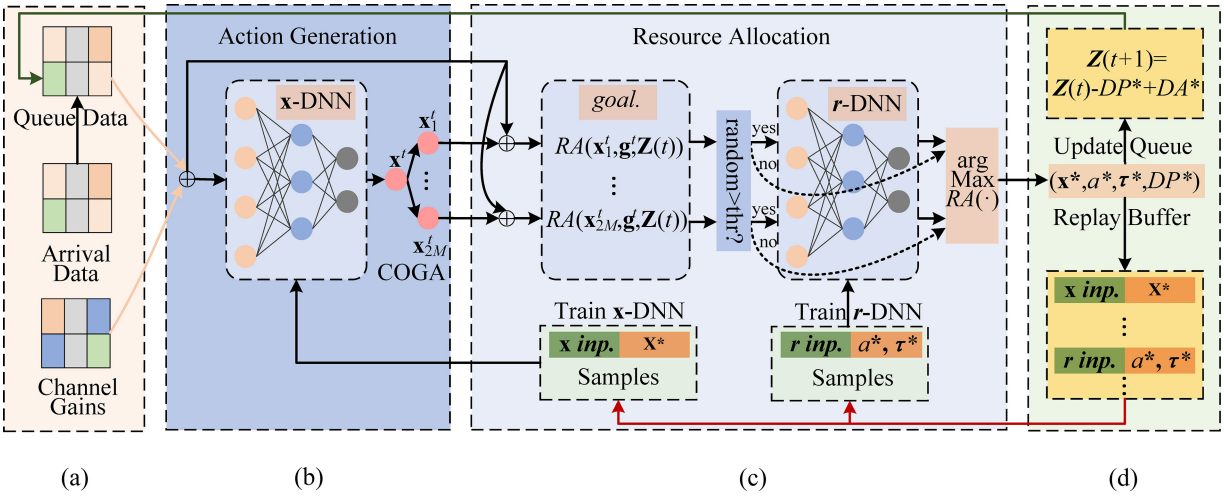



Fig. 2. QDRL overview. In (a), the obtained environmental information is used as the data set, which is the input of QDRL. In (b), the Actor module is leveraged to generate a series of potential offloading actions. In (c), the Critic module evaluates the outcomes produced by the Actor module and determines the optimal offloading action along with its corresponding resource allocation. In (d), according to the results of the Critic module, QDRL updates the task queue and updates the replay buffer. Besides, the QDRL trains **x**-DNN and ***r***-DNN at regular intervals.

TABLE I
SUMMARY OF NOTATIONS

| Notation | Description |
|---|---|
| $N$ | The number of wireless sensors |
| $S$ | The set of wireless sensors |
| $DA_i$ | The data arrival rate of sensor $s_i$ |
| $E^t_{har_i}$ | The sensor $s_i$ harvests energy at the $t$-the time frame |
| $E^t_{con_i}$ | The energy consumption of sensor $s_i$ at the $t$-the time frame |
| $DP_{L,i}$ | The amount of data processed in the sensor $s_i$ |
| $DP_{O,i}$ | The amount of data processed in the edge server |
| $Z_i(t)$ | The task queue length of sensor $s_i$ |
| $A_1, A_2$ | The accuracy of neural network |
| $g_i$ | The gain of the wireless channel between AP and sensor $s_i$ |
| $\mu$ | The efficiency of energy harvesting |
| $P$ | The transmit power of the AP's radio frequency energy |
| $\alpha$ | The percentage coefficient of the AP broadcasting energy time |
| $k_i$ | The coefficient of computation energy efficiency |
| $f_i$ | The processor chip's computing frequency |
| $\phi$ | The number of computation cycles needed to process one bit of raw data |
| $\tau_{L,i}$ | The percentage coefficient of sensor $s_i$ operation time |
| $B$ | The bandwidth between any one sensor and the AP |
| $v_u$ | The communication overhead |
| $N_0$ | The noise power |
| $x_i$ | The decision variable of sensor $s_i$ |
| $\gamma$ | The penalty factor |
| $w_i$ | The weight of sensor $s_i$ |

the task queue (a system is deemed stable if all its task queues maintain stability [18]). Moreover, we define $\mathbf{x}^t = \{x^t_1, \ldots, x^t_N\}$ and $\boldsymbol{\tau}^t = \{\tau^t_{O,1}, \ldots, \tau^t_{O,N}\}$ and assume $\mathbf{x} = \{\mathbf{x}^t\}^K_{t=1}$ and $\boldsymbol{\tau} = \{\boldsymbol{\tau}^t\}^K_{t=1}$. A summary of the notations is provided in Table I.

## IV. Problem Transformation and Algorithm Design

The optimization problem presented in (15) is an M-SMIP. This makes it challenging to solve directly using traditional RL algorithms. To this end, we propose a novel computation offloading framework based on DRL, namely, QDRL. This framework is designed to enhance the CR and ensure the system operates with long-term stability while minimizing CR losses. First, we employ the Lyapunov optimization technique to convent the optimization problem into a per-frame deterministic problem. Then, we design the DNN model and the COGA to generate offloading actions. Finally, the mathematical reasoning and learning-based methods are fused to achieve reasonable resource allocation. The framework of QDRL is presented in Fig. 2.

### A. Lyapunov-Based Problem Transformation

The M-SMIP problem is challenging to solve. To overcome this, we employ the Lyapunov optimization technique to convert this problem into a per-frame deterministic problem [18], [27]. Specifically, we denote $\boldsymbol{Z}(t) = \{Z_i(t)\}^N_{i=1}$ as the sum task queue of all sensors. Subsequently, the Lyapunov function $L(\boldsymbol{Z}(t))$ is introduced, as follows:

$$L(\boldsymbol{Z}(t)) = \frac{1}{2}\sum_{i=1}^{N} Z_i(t)^2. \tag{16}$$

Given $\boldsymbol{Z}(t)$, the one-shot Lyapunov drift $\Delta L(\boldsymbol{Z}(t))$ is modeled as

$$\Delta L(\boldsymbol{Z}(t)) = \mathbb{E}\big[L(\boldsymbol{Z}(t+1)) - L(\boldsymbol{Z}(t))|\boldsymbol{Z}(t)\big]. \tag{17}$$

Simultaneously squaring both sides of (14), we have

$$Z_i(t+1)^2 = Z_i(t)^2 + 2Z_i(t)\big(\mathrm{DA}^t_i - \mathrm{DP}^t_i\big) + \big(\mathrm{DA}^t_i - \mathrm{DP}^t_i\big)^2. \tag{18}$$

By subtracting $Z_i(t)^2$ from both sides of (18) and dividing by 2. Next, summing each term, we have

$$\begin{aligned}&\frac{1}{2}\sum_{i=1}^{N} Z_i(t+1)^2 - \frac{1}{2}\sum_{i=1}^{N} Z_i(t)^2\\ &= \frac{1}{2}\sum_{i=1}^{N}\big(\mathrm{DA}^t_i - \mathrm{DP}^t_i\big) + \sum_{i=1}^{N} Z_i(t)\big(\mathrm{DA}^t_i - \mathrm{DP}^t_i\big).\end{aligned} \tag{19}$$

Substituting (16) and (19) into (17), we have

$$\begin{aligned}\Delta L(\boldsymbol{Z}(t)) = &\frac{1}{2}\sum_{i=1}^{N}\mathbb{E}\Big[\big(\mathrm{DA}^t_i - \mathrm{DP}^t_i\big)^2\Big]\\ &+ \sum_{i=1}^{N}\mathbb{E}\big[Z_i(t)\big(\mathrm{DA}^t_i - \mathrm{DP}^t_i\big)\big].\end{aligned} \tag{20}$$

The first term on the right side of (20) satisfies

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{N}\mathbb{E}\Big[\big(\mathrm{DA}_i^t-\mathrm{DP}_i^t\big)^2\Big] &\le \frac{1}{2}\sum_{i=1}^{N}\mathbb{E}\Big[\big(\mathrm{DA}_i^t\big)^2+\big(\mathrm{DP}_i^t\big)^2\Big]\\ &\le \frac{1}{2}\sum_{i=1}^{N}\Bigg(\big(\mathrm{DA}_i^t\big)^2+\Bigg[T\max\Bigg\{\Big(\frac{\mu P g_i^t}{k_i}\Big)^{\frac{1}{3}}/\phi\\ &\qquad \mathbb{E}\Big[\frac{B}{v_u}\log_2\Big(1+\frac{P_i^{*t}g_i^t}{N_0}\Big)\Big]\Bigg\}\Bigg]^2\Bigg)\\ &= C \end{aligned} \tag{21}$$

where $DA_i^t$ is the data (i.e., task) arrival rate, which is given. $(\mu P g_i^t/k_i)^{(1/3)}/\phi$ represents the maximum CR of the sensor $s_i$, which is a constant. $\mathbb{E}[B/v_u\log_2(1+P_i^{*t}g_i^t/N_0)]$ denotes the maximum average transmission rate of the sensor $s_i$, which is a constant and $P_i^{*t}=P$. Therefore, $C$ is a constant. The upper bounded of $\Delta L(\mathbf{Z}(t))$ is

$$\Delta L(\mathbf{Z}(t)) \le C+\sum_{i=1}^{N}\mathbb{E}\Big[Z_i(t)\big(DA_i^t-\mathrm{DP}_i^t\big)\Big]. \tag{22}$$

We employ the drift-plus-penalty minimization method aimed at maximizing the RCR, while concurrently stabilizing the queue $\mathbf{Z}(t)$ during the $t$th time frame, as follows:

$$\Delta L(\mathbf{Z}(t))-V\cdot\sum_{i=1}^{N}\mathbb{E}\Big[w_i\mathrm{RCR}_i^t|\mathbf{Z}(t)\Big]. \tag{23}$$

Substituting (22) into (23), we have

$$C+\sum_{i=1}^{N}\mathbb{E}\Big[Z_i(t)\big(DA_i^t-\mathrm{DP}_i^t\big)\Big]-V\cdot\sum_{i=1}^{N}\mathbb{E}\Big[w_i\mathrm{RCR}_i^t|\mathbf{Z}(t)\Big] \tag{24}$$

where $V>0$ represents a scaling factor for the penalty. $\sum_{i=1}^{N}\mathbb{E}[w_i\mathrm{RCR}_i^t|\mathbf{Z}(t)]$ is a nonnegative number. Therefore, (24) also satisfies the inequality of (22).

In this article, we assume the duration $T=1$. Therefore, we have $\mathrm{CR}_i^t=\mathrm{DP}_i^t$ based on (10). By eliminating the constant terms (namely, $C$ and $\mathrm{DA}_i^t$) from (24) during the $t$th time frame, the QDRL can obtain the optimal offloading action through maximizing

$$\sum_{i=1}^{N}[Z_i(t)+V\cdot w_i]CR_i^t-V\cdot ERR_i^t \tag{25}$$

where $CR_i^t$ and $ERR_i^t$ are given in (11) and (13), respectively. We can easily know that the CR is not only affected by the sensor node weights but also by the CR losses. So far, we have obtained the per-frame deterministic problem with the Lyapunov optimization technique at the $t$th time frame, as follows:

$$\underset{\mathbf{x}^t,\alpha^t,\boldsymbol{\tau}^t}{\text{maximize}}\ \sum_{i=1}^{N}[Z_i(t)+V\cdot w_i]\mathrm{CR}_i^t-V\cdot\mathrm{ERR}_i^t \tag{26}$$

$$\text{subject to}\ \ CR_i^t\le Z_i(t)\quad \forall i\in N \tag{26a}$$

$$\sum_{i=1}^{N}\tau_{O,i}^t+\alpha^t\le 1 \tag{26b}$$

$$\alpha^t\ge 0,\tau_{O,i}^t\ge 0\quad \forall i\in N \tag{26c}$$

$$x_i^t\in\{0,1\}\quad \forall i\in N \tag{26d}$$

where the constraints of (26a)–(26d) are the same as (15a)–(15e).

### *B. Actor: Deep Learning and Genetic Algorithm-Based Offloading Action Generation*

It is well known that deep learning models (e.g., CNN [34] and LSTM [35]) have achieved great success in prediction. In this section, we design the **x**-DNN model in Fig. 2(b) to predict the computation offloading action of all sensors. Specifically, the optimal mapping function $f_{\delta^t}(\cdot)$ of an approximation offloading action $\mathbf{x}^t$ as

$$\mathbf{x}^t=f_{\delta^t}\Big(\mathbf{g}^t,\mathbf{Z}(t)\Big) \tag{27}$$

where $f_{\delta^t}(\cdot)$ is learned from $\mathbf{g}^t$ and $\mathbf{Z}(t)$). $\mathbf{g}^t=\{g_i^t\}_{i=1}^N$, $\mathbf{Z}(t)=\{Z_i(t)\}_{i=1}^N$. $g_i^t$ and $Z_i(t)$ represent the channel gain and task queue of the sensor $s_i$ at the $t$th time frame. $\delta^t$ is the trainable weight parameter. Similar to [26], we also apply the Adam algorithm [36] to update the $\delta^t$, as

$$\begin{aligned}\mathrm{LOSS}_{\delta^t}=-\frac{1}{|\epsilon^t|}\sum_{\varepsilon\in\epsilon^t}\Big(&\big((\mathbf{x}^*)^{\varepsilon}\big)^{\mathsf{T}}\log f_{\delta^t}\big(\mathbf{g}^{\varepsilon},\mathbf{Z}(\varepsilon)\big)\\ &+\big(1-(\mathbf{x}^*)^{\varepsilon}\big)^{\mathsf{T}}\log\big(1-f_{\delta^t}\big(\mathbf{g}^{\varepsilon},\mathbf{Z}(\varepsilon)\big)\big)\Big)\end{aligned} \tag{28}$$

where $|\epsilon^t|$ represents the size of $\epsilon^t$, and the $\mathsf{T}$ is the transpose operator. $(\mathbf{g}^{\varepsilon},\mathbf{Z}(\varepsilon))$ is a training data sample from the memory, represented by a set of time index $\epsilon^t$, where $\varepsilon\in\epsilon^t$.

This article is implemented based on the binary offloading mechanism, but $\mathbf{x}^t$ usually be any value between (0,1). Therefore, we need to quantize $\mathbf{x}^t$, i.e., satisfy $x_i^t=0$ or 1. Although the traditional scheme can obtain the best offloading action from $2^N$ offloading actions ($N$ is the number of sensors), it will face very high computational complexity [26], [37]. In QDRL, we can obtain an approximation of the offloading action by $f_{\delta^t}(\cdot)$ based on the proper activation function, e.g., *ReLu* [38], *Sigmoid* [39], and *tanh* functions [40]. However, it is difficult for the above schemes to generate a high-quality offloading action. Inspired by the theory of evolution (natural selection and survival of the fittest), we apply the development of populations to break this bottleneck. With this in mind, we introduce a computation offloading quantization method based on genetic algorithm (COGA) to generate $M^t$ (we set $M^t=N$) offloading actions about $\mathbf{x}^t$, which can balance the performance and complexity. The schematic of COGA is shown in Fig. 3. The COGA includes three steps.

1) *Natural Selection:* COGA first selects all offloading actions generated by the order-preserving quantization (OP) [5] method as the population (pop), as follows:

$$\mathrm{pop}=\Big\{\breve{\mathbf{x}}_1^t,\breve{\mathbf{x}}_2^t,\ldots,\breve{\mathbf{x}}_M^t\Big\}. \tag{29}$$

2) *Crossover:* The $i$th individual crosses with the $j$th individual with a probability $CP$ to achieve the gene change

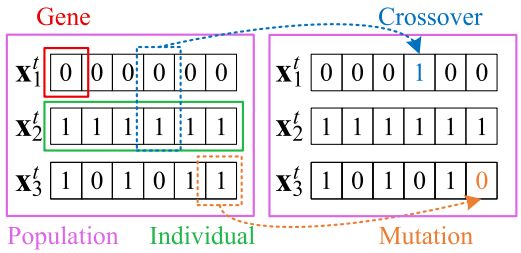



Fig. 3. Schematic of COGA. The example shows the computing mode of six sensors with a population of 3. The "Gene" signifies a sensor's computing mode, either local or edge computing. The "Individual" denotes an offloading action of all sensors. The "Population" represents a set of candidate offloading actions. The crossover between individuals $\mathbf{x}_1^t$ and $\mathbf{x}_2^t$ in the population achieves the change of $\mathbf{x}_1^t$ at the fourth gene. In addition, individual $\mathbf{x}_3^t$ has a mutation at the last gene.

at the point position, as follows:

$$\text{pop}[i, \text{point}] = \text{pop}[j, \text{point}]. \tag{30}$$

3) *Mutation:* The point gene of the $i$th individual is mutated with a probability $MP$, as follows:

$$\text{pop}[i, \text{point}] = 1 \quad \text{if} \quad \text{pop}[i, \text{point}] = 0 \quad \text{else} \quad 0. \tag{31}$$

After crossover and mutation, we can get a new population. Next, we merge the old population with the new population to obtain a population that contains more offloading actions, as follows:

$$\Phi^t = \text{pop}||\left\{\check{\mathbf{x}}_1^t, \check{\mathbf{x}}_2^t, \ldots, \check{\mathbf{x}}_M^t\right\} \tag{32}$$

where $\Phi^t = \{\mathbf{x}_l^t | \mathbf{x}_l^t \in \{0,1\}^N, l = 1, 2, \ldots, 2M^t\}$ is total population and $||$ represents concatenate operation.

Finally, we employ the resource allocation algorithm to evaluate $\Phi^t$ and identify the optimal offloading action $\mathbf{x}^t$, as follows:

$$\mathbf{x}^t = \arg\max_{\mathbf{x}_l^t \in \Phi^t} RA\left(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t)\right). \tag{33}$$

where $RA(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t))$ is a resource allocation function, which can be solved by giving $\mathbf{x}_l^t$, $\mathbf{g}^t$, and $\mathbf{Z}(t)$. We will detail how $RA(\cdot)$ is solved in Section IV-C.

### C. Critic: Mathematical Reasoning and Deep-Learning-Based Resource Allocation

In Section IV-B, we obtain the candidate set of offloading action $\Phi^t$ at the $t$th time frame. For a given offloading action $\mathbf{x}_l^t \in \Phi^t$, substituting (11) and (12) into (26), and the total RCR is rewritten as follows:

$$\begin{aligned}\underset{\alpha,\boldsymbol{\tau}}{\text{maximize}} \; & \sum_{i\in X_0}\left(C1\cdot q_i\left(\frac{g_i}{k_i}\right)^{\frac{1}{3}}\alpha^{\frac{1}{3}} - V\cdot(1-A_1)\text{DA}_i\gamma\right) \\ & + \sum_{j\in X_1}\left(C2\cdot q_j\tau_{O,j}\ln\left(1+C3\cdot\frac{g_j^2\alpha}{\tau_{O,j}}\right)\right. \\ & \left. - V\cdot(1-A_2)\text{DA}_j\gamma\right)\end{aligned} \tag{34}$$

where the index set of sensors with $x_i^t = 1$ is represented as $X_1^t$, and the complementary set of sensors is denoted by $X_0^t$. $q_i = [Z_i(t) + V\cdot w_i]$, $C1 = (uP)^{(1/3)}/\phi$, $C2 = B/(V_u \ln 2)$, $C3 = \mu P/N_0$.

Subsequently, we design an efficient and accurate resource allocation algorithm based on mathematical reasoning and deep learning to obtain the resource allocation of the offloading action $\mathbf{x}_l^t$ and stabilizing task queues for all sensors. This algorithm consists of two parts: 1) mathematical reasoning scheme, indicated by the dashed arrows in Fig. 2(c) and 2) deep learning scheme, indicated by the $\boldsymbol{r}$-DNN in Fig. 2(c). The content is as follows.

*1) Resource Allocation Scheme Based on Mathematical Reasoning:* When $\mathbf{x}_l^t \in \Phi^t$ is given, $X_0^t$ and $X_0^t$ are known. We first apply the Lagrangian multiplier to solve the optimization problem of (34), as follows:

$$\begin{aligned}L(\alpha,\boldsymbol{\tau},z) = & \sum_{i\in X_0}\left(C1\cdot q_i\left(\frac{g_i}{k_i}\right)^{\frac{1}{3}}\alpha^{\frac{1}{3}} - V\cdot(1-A_1)DA_i\gamma\right) + \\ & \sum_{j\in X_1}\left(C2\cdot q_j\tau_{O,j}\ln\left(1+C3\cdot\frac{g_j^2\alpha}{\tau_{O,j}}\right) - V\cdot(1-A_2)\text{DA}_j\gamma\right) \\ & - z\left(\alpha + \sum_{j\in X_1}\tau_{O,j} - 1\right).\end{aligned} \tag{35}$$

Then, we use the idea of Primal–Dual to solve the problems involved in (35). The Primal problem is

$$F(z) = \underset{a,\boldsymbol{\tau}}{\text{maximize}}\left\{L(a,\boldsymbol{\tau},z) | \alpha \geq 0, \tau_{O,j} \geq 0, j \in X_1\right\}. \tag{36}$$

The corresponding Dual problem is

$$\underset{z}{\text{minimize}}\{F(z) | z \geq 0\}. \tag{37}$$

If both (36) and (37) are satisfied at the same time, then there is a set of solutions $(\alpha^*, \boldsymbol{\tau}^*, z^*)$ such that the first derivative of $L(\alpha, \boldsymbol{\tau}, z)$ is 0, i.e.,

$$\begin{aligned}\left.\frac{\mathrm{d}L}{\mathrm{d}\tau_{O,j}}\right|_{\alpha=\alpha^*,\tau_{O,j}=\tau_{O,j}^*,z=z^*} = & \; C2\cdot q_i\ln\left(1+C3\cdot\frac{g_j^2\alpha^*}{\tau_{O,j}^*}\right) \\ & - \frac{C2\cdot C3\cdot q_j g_j^2\alpha^*\tau_{O,j}^{*-1}}{1+C3\cdot g_j^2\alpha\tau_{O,j}^{*-1}} - z^* = 0\end{aligned} \tag{38}$$

$$\begin{aligned}\left.\frac{\mathrm{d}L}{\mathrm{d}\alpha}\right|_{\alpha=\alpha^*,\tau_{O,j}=\tau_{O,j}^*,z=z^*} = & \; C1\cdot\frac{1}{3}(\alpha^*)^{-\frac{2}{3}}\sum_{i\in X_0}q_i\left(\frac{g_i}{k_i}\right)^{\frac{1}{3}} \\ & + \sum_{j\in X_1}\frac{C2\cdot C3\cdot q_j g_j^2}{1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}} - z^* = 0.\end{aligned} \tag{39}$$

According to (38), we have

$$\begin{aligned}& C2\cdot q_i\ln\left(1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}\right) \\ & = \frac{C2\cdot C3\cdot q_j g_j^2\alpha^*\tau_{O,j}^{*-1}}{1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}} + z^*.\end{aligned} \tag{40}$$

Dividing both sides of (40) by $C2\cdot q_i$, we have

$$\begin{aligned}\ln\left(1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}\right) = & \left(1+\frac{z^*}{C2\cdot q_j}\right) \\ & - \frac{1}{1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}}.\end{aligned} \tag{41}$$

Adding $(1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1})^{-1}$ and taking exponential at both sides, we have

$$\left(1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}\right)\exp\left(\frac{1}{1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}}\right)=\exp\left(1+\frac{z^*}{C2\cdot q_j}\right). \tag{42}$$

When $\beta>0$ and $\theta>0$ that satisfy $(1/\beta)\exp(\beta)=\theta$, it holds that $-\beta\exp(-\beta)=-(1/\theta)$. So, we have $\beta=-\mathrm{LW}(-[1/\theta])$, where $\mathrm{LW}(z^*)$ denotes the Lambert-W function [5]. According to (42) and the Lambert-W function $\mathrm{LW}(z^*)$, we derive the following:

$$\frac{1}{\left(1+C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}\right)}=-\mathrm{LW}\left(-\frac{1}{\exp\left(1+\frac{z^*}{C2\cdot q_j}\right)}\right). \tag{43}$$

Simplifying (43), we have

$$\frac{\tau_{O,j}^*}{\alpha^*}=C3\cdot g_j^2F_{o,j}(z^*) \tag{44}$$

where $F_{O,j}(z^*)=[-(\mathrm{LW}(-1/\exp(1+[z^*/C2\cdot q_j])))^{-1}-1]^{-1}$. Multiplying both sides of (44) by $\alpha^*$, we have

$$\tau_{O,j}^*=C3\cdot g_j^2F_{O,j}(z^*)\alpha^*. \tag{45}$$

When $\sum_{j\in X_1^t}\tau_{O,j}^*+\alpha^*=1$, $(\alpha^*,\boldsymbol{\tau}^*,z^*)$ holds at the optimal solution, substituting $\sum_{j\in X_1^t}\tau_{O,j}^*+\alpha^*=1$ into (45), we have

$$\alpha^*=\frac{1}{1+C3\cdot\left(\sum_{j\in X_1}g_j^2F_{O,j}(z^*)\right)}=F_C(z^*). \tag{46}$$

Substituting (46) into (39), we have

$$C1\cdot\frac{1}{3}\left(F_C(z^*)\right)^{-\frac{2}{3}}\sum_{i\in X_0}q_i\left(\frac{g_i}{k_i}\right)^{\frac{1}{3}}+\sum_{j\in X_1}\frac{C2\cdot C3\cdot q_jg_j^2}{1+\frac{1}{F_{O,j}(z^*)}}-z^*=0 \tag{47}$$

where $(F_{O,j}(z^*))^{-1}=C3\cdot g_j^2\alpha^*\tau_{O,j}^{*-1}$ based on (45). We apply the bisection search method proposed in [41] to solve (47), then, the solutions of (45) and (46) can be obtained, which constitute the optimal resource allocation of the computation offloading action $\mathbf{x}_l^t$.

*2) Resource Allocation Scheme Based on Deep Learning:* In (34), $\alpha$ and $\boldsymbol{\tau}$ satisfy the constraints of (26b), especially, $\alpha+\sum_{j\in X_1}\tau_{O,j}=1$ also exists, where $\boldsymbol{\tau}=\sum_{j\in X_1}\tau_{O,j}$. It is not difficult to see that the solution of $\alpha$ and $\boldsymbol{\tau}$ is a combinatorial optimization problem. This problem is not easy to solve with traditional methods, but it is easy to obtain approximate solutions for deep learning [42]. To obtain a solution for (34) quickly, this article considers using the deep learning model to obtain $\alpha$ and $\boldsymbol{\tau}$, simultaneously.

*Solution for $\alpha$ and $\boldsymbol{\tau}$:* We design the deep learning model $\boldsymbol{r}$-DNN in Fig. 2(c) to predict $\alpha$ and $\boldsymbol{\tau}$, and the corresponding mapping function is $f_{\psi^t}$, as follows:

$$\boldsymbol{r}^t=f_{\psi^t}\left(\mathbf{g}^t,\mathbf{Z}(t),\mathbf{x}_l^t\right) \tag{48}$$

where $\boldsymbol{r}^t$ is the solution formed by $\alpha$ and $\boldsymbol{\tau}$. $\psi^t$ is a trainable parameter. Different from $f_{\delta^t}$ in (27), the input of $f_{\psi^t}$ includes the channel gain $\mathbf{g}^t$ and the task queue of all sensors $\mathbf{Z}(t)$ in addition to the offloading decision action $\mathbf{x}_l^t$. Similar to $\mathbf{x}$-DNN, the Adam algorithm [36] is also used to update $\psi^t$ in $\boldsymbol{r}$-DNN, as follows:

$$\mathrm{LOSS}_{\psi^t}=-\frac{1}{|\epsilon^t|}\sum_{\varepsilon\in\epsilon^t}\Big(\left((\boldsymbol{r}^*)^{\varepsilon}\right)^{\mathsf{T}}\log f_{\psi^t}\left(\mathbf{g}^{\varepsilon},\mathbf{Z}(\varepsilon),\mathbf{x}_l^{\varepsilon}\right)+\left(1-(\boldsymbol{r}^*)^{\varepsilon}\right)^{\mathsf{T}}\log\left(1-f_{\psi^t}\left(\mathbf{g}^{\varepsilon},\mathbf{Z}(\varepsilon),\mathbf{x}_l^{\varepsilon}\right)\right)\Big) \tag{49}$$

where $\varepsilon$ and $\epsilon^t$ are the same as the parameters of the same name of (28). In the above way, we can obtain $\alpha$ and $\boldsymbol{\tau}$, which constitute the optimal resource allocation solution of the computation offloading action $\mathbf{x}_l^t$.

*No Free Lunch Theorem [43]:* The method of mathematical reasoning can accurately obtain reasonable resource allocation, but it takes a lot of computation time, which is difficult to meet real-time requirements. Although the method based on deep learning is very fast, $\boldsymbol{r}$-DNN is trained with bad input and label pairs, which results in the results of the inference being usually random and bad. It is difficult for $\boldsymbol{r}$-DNN to obtain a suitable solution. This article aims to effectively combine mathematical reasoning-based and deep-learning-based solutions to obtain an efficient and accurate solution. Specifically, the method based on mathematical reasoning participates in resource allocation with the probability of *thr*, whose results are used as a part of the training sample of $\boldsymbol{r}$-DNN. The learning-based schemes participate in resource allocation with $(1-thr)$ probability, whose results also are used as another part of the training sample of $\boldsymbol{r}$-DNN. After introducing *thr*, the QDRL does not need to perform mathematical reasoning every time, and it can also obtain better training samples for $\boldsymbol{r}$-DNN. The $\boldsymbol{r}$-DNN trained in this way can be both fast and accurate in the inference, thereby improving the performance of QDRL.

Overall, the Actor and Critic continue to learn based on the replay buffer, and as the training progresses, the former generates better offloading decisions and the latter obtains more and more accurate evaluation scores. QDRL continuously improves the offloading policy under the AC RL mechanism until convergence. We provide the pseudocode of the QDRL algorithm in Algorithm 1. Algorithm 1 accepts the channel gain $\mathbf{g}^t$ and data arrival rate $DA^t$ as the input, aiming to output the optimal offloading action $\mathbf{x}^*$ and the resource allocation solution $(\alpha^*,\boldsymbol{\tau}^*)$. Here, $\mathbf{g}^t$ is the channel gain between all wireless sensors and the AP and $DA^t$ is the data arrival rate on all wireless sensors. We initialize $\mathbf{x}$-DNN and $\boldsymbol{r}$-DNN with random parameters $\delta^t$ and $\psi^t$, respectively (line 1) and set the task queue initial value of each sensor to 0 (line 2). In the test, Algorithm 1 continues to iterate $K$ rounds (line 3).

When new data (i.e., new tasks) are generated, Algorithm 1 first updates the task queue $Z_i(t)$ of the sensor based on (14) (line 4). Next, the Actor starts. Specifically, Algorithm 1 uses $\mathbf{x}$-DNN to produce a preliminary offloading action

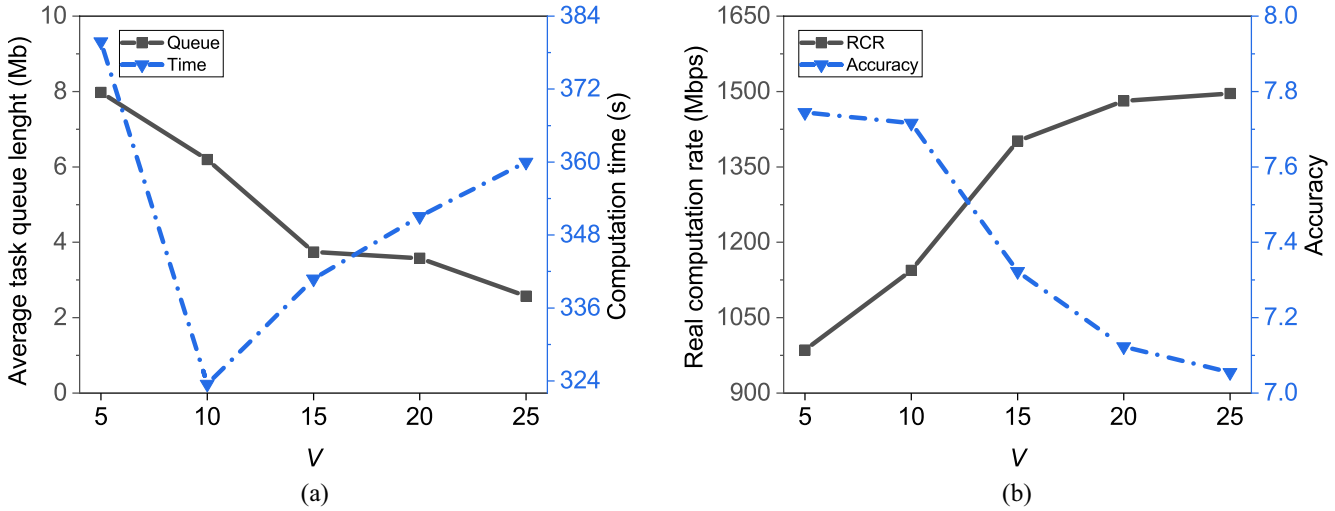



Fig. 4. Impact of $V$ on QDRL. (a) Impact of $V$ on the average task queue length and computation time. (b) Impact of $V$ on the RCR and accuracy.

**Algorithm 1** QDRL Algorithm

**Input:**
Channel gain $\mathbf{g}^t$ and data arrival rate $DA^t$
**Output:**
Offloading action $\mathbf{x}^*$ and its resource allocation $(\alpha^*, \boldsymbol{\tau}^*)$

1: Initialize the $\mathbf{x}$-DNN and $\boldsymbol{r}$-DNN with random parameters $\delta^t$ and $\psi^t$ respectively;
2: Initialize the task queue $Z_i(1) = 0$, where $i = 1, 2, \ldots, N$;
3: **for** $t$ in $(1, K)$ **do**
4: Update the $Z_i(t)$ based on the Equation (14);
5: Produce a preliminary offloading action $\mathbf{x}^t$ using the $\mathbf{x}$-DNN based on the Equation (27);
6: Generate $2M$ candidate offloading actions related to $\mathbf{x}^t$ with COGA;
7: **if** *random* $>$ *thr* **then**
8: Solve the function $RA\left(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t)\right)$ with $\boldsymbol{r}$-DNN;
9: **else**
10: Solve the function $RA\left(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t)\right)$ with mathematical reasoning;
11: **end if**
12: Determine the optimal offloading action and its corresponding resource allocation through $\mathbf{x}^* = \arg\max_{\mathbf{x}_l^t \in \Phi^t} RA\left(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t)\right)$;
13: Update the replay buffer by adding $\left(\left(\mathbf{g}^t, \mathbf{Z}(t)\right), \mathbf{x}^*\right)$ and $\left(\left(\mathbf{g}^t, \mathbf{Z}(t), \mathbf{x}^*\right), \boldsymbol{r}^*\right)$, where $\boldsymbol{r}^t$ is formed by $\alpha^*$ and $\boldsymbol{\tau}^*$;
14: **if** $t$ mod $10 = 0$ **then** // "10" represents the training interval
15: Randomly sample a batch of data set $\left(\left(\mathbf{g}^\varepsilon, \mathbf{Z}(\varepsilon)\right), (\mathbf{x}^*)^\varepsilon\right)$ and $\left(\left(\mathbf{g}^\varepsilon, \mathbf{Z}(\varepsilon), (\mathbf{x}^*)^\varepsilon\right), (\boldsymbol{r}^*)^\varepsilon\right)$ from the memory, where $\varepsilon \in \epsilon^t$;
16: Train the $\mathbf{x}$-DNN and $\boldsymbol{r}$-DNN with $\left(\left(\mathbf{g}^\varepsilon, \mathbf{Z}(\varepsilon)\right), (\mathbf{x}^*)^\varepsilon\right)$ and $\left(\left(\mathbf{g}^\varepsilon, \mathbf{Z}(\varepsilon), (\mathbf{x}^*)^\varepsilon\right), (\boldsymbol{r}^*)^\varepsilon\right)$, and update $\delta^t$ and $\psi^t$ with the Adam algorithm, where $\varepsilon \in \epsilon^t$;
17: **end if**
18: **end for**

$\mathbf{x}^t$ (line 5) and applies COGA to generate $2M$ candidate offloading actions related to $\mathbf{x}^t$ (line 6). Before starting the Critic, Algorithm 1 needs to judge the relationship between the *random* and *thr*, aiming to choose which method to evaluate the actions generated by the Actor. If *random* $>$ *thr*, Algorithm 1 selects $\boldsymbol{r}$-DNN to solve $RA(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t))$ (line 8). Otherwise (line 9), Algorithm 1 uses mathematical reasoning to solve $RA(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t))$ (line 10). Subsequently, determining the optimal offloading action and its corresponding resource allocation by maximizing $RA(\mathbf{x}_l^t, \mathbf{g}^t, \mathbf{Z}(t))$ (line 12), which are used as samples to update the replay buffer. Finally, Algorithm 1 randomly selects a batch of data set from memory for training $\mathbf{x}$-DNN and $\boldsymbol{r}$-DNN every ten rounds and updates $\delta^t$ and $\psi^t$ with Adam (lines 14–17).

## V. Experiments

In the following section, we provide a detailed description of the experimental setup and the results derived from it. The results obtained with QDRL are compared to five baseline algorithms for computation offloading. QDRL is implemented with Pytorch and optimized by Adam. All the results are evaluated on an Intel Xeon Silver 4114 2.2-GHz CPU and 13 GB of memory. The Python program implementation of QDRL is available in [44].

### A. Experimental Setup

*1) Data Set Description and Parameter Settings:* The data set used in this article includes the data arrival rate $DA_i$ and channel gain $g_i$. The task data arrivals of all the sensors follow the exponential distribution with an equal average rate, i.e.,

$$\mathbb{E}[\mathrm{DA}_i] = \lambda_i, i = 1, \ldots, N \tag{50}$$

where $\lambda_i = 3$ Mb/s. Similar to [5], we assume that the channel gain is $g_i = \overline{g}_i \chi$, which obeys the rayleigh fading channel model. Where $\overline{g}_i$ is the average channel gain and $\chi = 0.3$ represents an independent exponential random variable of unit mean. In particular, $\overline{g}_i$ is expressed as

$$\overline{g}_i = A_d \left(\frac{3 \cdot 10^8}{4\pi f_c d_i}\right)^{d_e}, i = 1, \ldots, N \tag{51}$$

where the antenna gain is represented by $A_d = 4.11$, $f_c = 915$ MHz signifies the carrier frequency, $d_i$ represents the distance

TABLE II
SIMULATION PARAMETERS (PARA) [26]

| Para | Value | Para | Value | Para | Value |
|---|---|---|---|---|---|
| $P$ | 3 W | $N_0$ | 1e-10 W | learning rate | 0.0009 |
| $\mu$ | 0.7 | $V_u$ | 1.1 | memory size | 1024 |
| $k_i$ | 1e-26 | $w_i$ | 1 | training interval | 10 |
| $V$ | 10 | $\lambda_i$ | 3 Mbps | $\|\epsilon\|$ | 256 |
| $\phi$ | 100 | $B$ | 2 MHz | $MP$ | 0.003 |
| $T$ | 1 s | $\gamma$ | 0.5 | $CP$ | 0.8 |

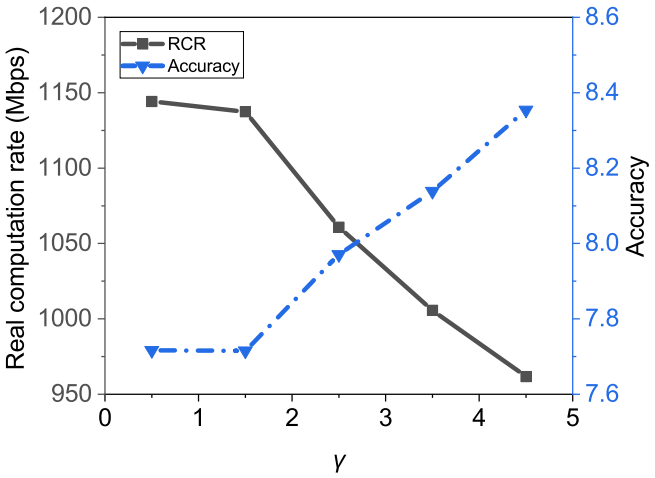



Fig. 5. Impact of $\gamma$ on QDRL performance.

between sensor $s_i$ and AP, and $d_e = 2.8$ stands for the path loss exponent. Both the x-DNN and r-DNN are fully connected deep neural networks, incorporating one input layer, two hidden layers, and one output layer. Specifically, the first and second hidden layers are equipped with 120 and 80 hidden neurons, respectively. The activation functions of the output layers of **x**-DNN and ***r***-DNN are *Sigmoid* and *Softmax*, respectively. The activation function of the other layers in both DNNs is *ReLU*. Other crucial parameters can be found listed in Table II.

*2) Baseline Algorithms:* This article demonstrates the advantages of QDRL by comparing QDRL with five baseline algorithms, i.e., LyCD [41], DROO [5], the classical AC, Edge-only, and Local-only. LyCD exchanges the computing mode of each sensor in each round. The iteration stops when the computing mode exchange cannot provide further CR performance. LyCD has been proven to achieve near-optimal performance under different $N$. However, it takes an intolerant time to obtain the solution. DROO is a short-sighted approach that maximizes the RCR per-frame but ignores the backlog of task queues that cause system instability. Similar to DROO, AC also causes queue backlogs and system instability. In contrast, AC performs faster because it generates only a small number of offloading decisions. Nevertheless, this makes it difficult to quickly obtain the optimal offloading decision. Edge-only represents all $N$ sensors offloading their tasks to the edge server, which increases the delay and decreases system revenue. Local-only represents all $N$ sensors that only perform local computation, which increases the sensor's task queue and causes system instability.

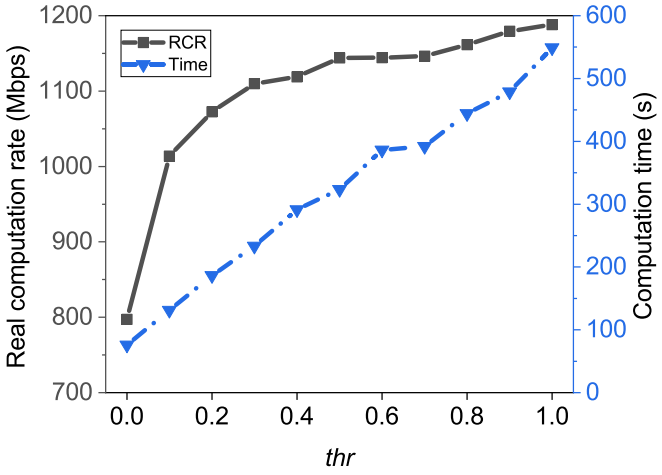



Fig. 6. Impact of *thr* on QDRL performance.

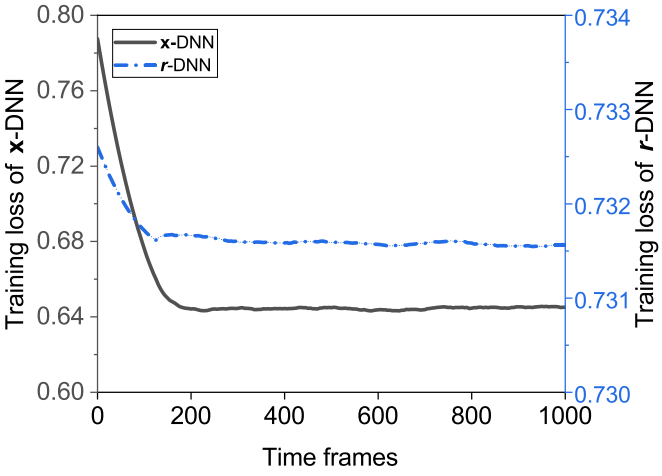



Fig. 7. Training loss for two deep neural networks from QDRL.

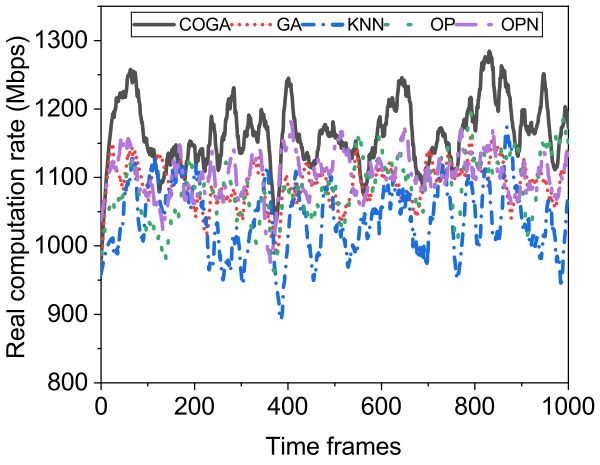



Fig. 8. Comparison of different offloading action quantization methods.

*3) Evaluation Metrics:* This article uses four commonly used metrics to evaluate QDRL, i.e., the computation time, the average task queue length, the RCR, and the normalized RCR. The smaller the computation time, the faster the algorithm execution speed. The smaller the average task queue length, the more stable the system. The higher the RCR, the

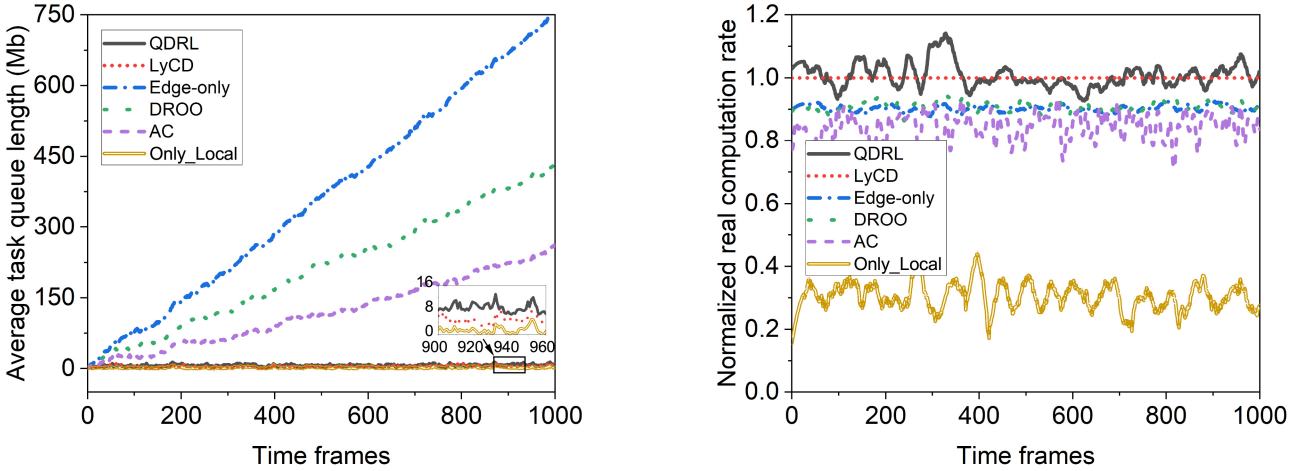



Fig. 9. Performance comparison under long-term continuous constraints. (a) Comparison of the average task queue length. (b) Comparison of the normalized RCR.

greater the system benefits. Through comprehensive simulations, we have confirmed that the LyCD method approaches optimal performance [41]. Therefore, we adopt LyCD as the performance benchmark for our QDRL method. To compare the performance of different schemes effectively, we define the normalized $\text{RCR} = \text{RCR}_{\text{method}}/\text{RCR}_{\text{LyCD}}$, where $\text{RCR}_{\text{LyCD}}$ represents the RCR of the LyCD algorithm and $\text{RCR}_{\text{method}}$ represents the RCR of other methods, e.g., QDRL, DROO, AC, Edge-only, and Local-only. The closer the value of the normalized RCR to 1, the better its performance.

## B. Results and Analysis

*1) Parameter Sensitivity:* Since different parameter settings will impact the performance of QDRL, we need to study the parameter sensitivity of QDRL. Here, we conduct experiments in consecutive 5000 time frames and vary a parameter at each time as well as set others to the default value, where $N = 10$ and $\lambda_i = 3$ Mb/s in (50). (By default, the following experiments are all done under this setting.) Specifically, we first vary the value of $V$ from 5 to 25. The results are reported in Fig. 4. As $V$ increases, the average queue length and RCR decrease gradually, while accuracy increases gradually. Unlike them, the computation time first decreases and then increases. Therefore, setting $V$ to 10 can achieve better performance. Then, we vary the value of penalty factor $\gamma$ from 0.5 to 4.5. The results are reported in Fig. 5. As the penalty factor $\gamma$ increases, the accuracy gradually increases while the RCR decreases gradually. Using $\gamma = 2.5$ is good enough to obtain the performance of QDRL. However, we hope to obtain a higher RCR, so set $\gamma$ to 0.5 in this article. Last but not least, we vary the value of *thr* from 0 to 1. The results are reported in Fig. 6. As *thr* increases, the RCR of QDRL first increases rapidly and then flattens out, but the computation time continues to increase rapidly. When $thr = 0.5$, we know that QDRL not only improves the benefit of QDRL but also reduces its time cost.

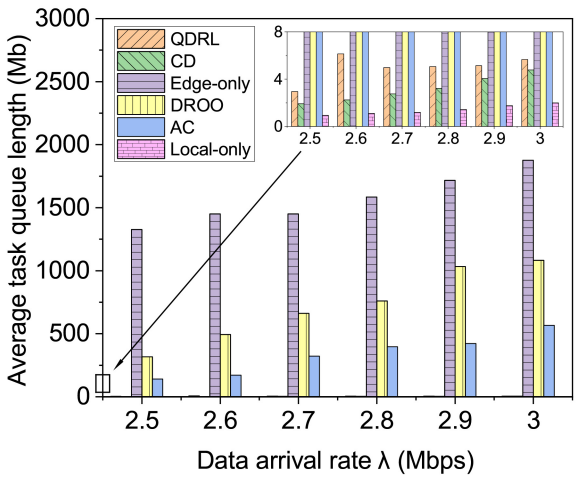
*2) Convergence of QDRL:* As shown in Fig. 7, we evaluate the convergence of two DNN models (i.e., **x**-DNN and ***r***-DNN) from QDRL with respect to the training episodes, where the learning rate is 0.0009. The two curves in the figure represent the training loss of **x**-DNN and ***r***-DNN. It is evident that losses decline with the increment of training episodes and eventually stabilize, reaching convergence in 200 time frames.

*3) Effects of COGA and thr on QDRL:* To further explore the effect of COGA and *thr* in our proposed scheme, we first introduce different offloading action quantization strategies (i.e., OP, noisy OP (NOP), and KNN [41]) in QDRL to compare with QDRL considering COGA, then analyze the effect of introducing parameter *thr* in QDRL. As shown in Fig. 8, the COGA scheme outperforms the other four methods in terms of RCR performance. COGA applies GA to improve OP, in which OP can obtain order-preserving offloading actions but is prone to falling into local optimum, and the introduced GA can break the bottleneck of OP and obtain a more global optimal solution.

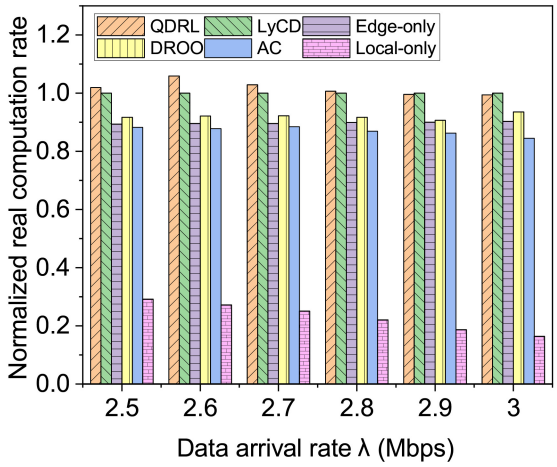
In Fig. 6, when $thr = 0$, QDRL only applies the deep neural network model to achieve resource allocation. $thr = 1$ means that QDRL completely uses a mathematical reasoning scheme to achieve resource allocation. The former approach merely requires leveraging the trained model for inference, thereby enabling the realization of resource allocation very quickly. However, the model ***r***-DNN used to implement inference resource allocation in RL is trained by poor data sets, which results in the inference results being relatively poor. The latter can directly obtain reasonable resource allocation based on mathematical reasoning, but repeatedly calling functions for calculation will consume a lot of time, and it poses a challenge in meeting the system's real-time requirements. This article introduces $thr = 0.5$ to realize the ingenious combination of the learning-based scheme and the mathematical reasoning-based scheme, which leads to a shorter time and higher RCR for QDRL to complete the same task.

*4) Performance Comparison:* We compare the results of QDRL with five baseline algorithms under different conditions. The content is as follows:

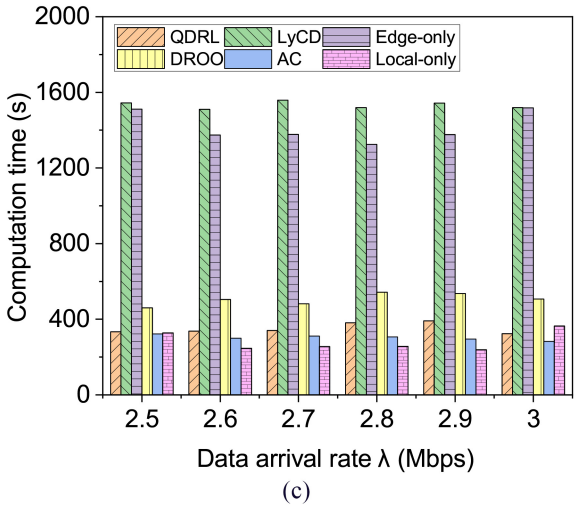
*Performance Comparison Under Long-Term Continuous Constraints:* As shown in Fig. 9, QDRL approaches and even




(a)




(b)




(c)

Fig. 10. Performance comparison under different data arrival rates. (a) Comparison of the average task queue length. (b) Comparison of the normalized RCR. (c) Comparison of the computation time.

exceeds near-optimal solution (i.e., LyCD) on the same data set using all metrics, demonstrating the advantage of our model. Specifically, QDRL achieves average task queue length and normalized RCR improvement in comparison with the excellent scheme (i.e., DROO). Moreover, we can make the following observations.

1) The average queue length and RCR of Local-only are very small, while the average queue length and RCR of Edge-only are large. From the perspective of the average queue length, this is because the former can quickly process data locally, while the latter causes a backlog of task queue time due to the inability to transmit data in a short time. The reason for anomalous RCRs is that they do not take edge–end collaboration into account.
2) The performance of DROO and AC in the average task queue length is between Edge-only and Local-only, and the performance in normalized RCR is similar to or even exceeds that of Edge-only. However, since they neglect the impact of the task queue, there is still a queue backlog in the system and the RCR cannot be significantly improved. Compared with them, QDRL not only considers edge–end collaboration but also considers the optimization of task queues and the maximization of long-term RCR benefits, so as to obtain near-optimal performance.

*Performance Comparison Under Different Data Arrival Rates:* In real-world scenarios, the generation and arrival of data (i.e., tasks) is random and dynamic. As depicted in Fig. 10, we test the performance of different methods by varying the data arrival rate. It is evident that the task queue of the sensor in Edge-only, DROO, and AC continues to accumulate as the data arrival rate increases. This is because they cannot process the arriving data in time. In contrast, the average task queue of Local-only is always low, because it only processes data locally, which is real time. Unlike them, as the data arrival rate increases, the average task queue of QDRL has no obvious increase and always keeps a relatively small range like LyCD. Its RCR can close to or even surpass the near-optimal algorithm LyCD. Besides, QDRL also has a great advantage in terms of computation time.

*Performance Comparison Under Different Minimum Tolerance Accuracy:* There are more and more tasks have high accuracy requirements in IIoT. To verify whether QDRL and its comparison schemes can be applied to these scenarios, we conduct extensive experiments under different minimum tolerance accuracy, as shown in Fig. 11. The results show that the average task queue of QDRL is still low under different minimum tolerance accuracy, indicating that the whole system is stable. In addition, QDRL is second only to the approximate optimal algorithm and outperforms other comparison schemes in normalized RCR, indicating that the performance of QDRL is advanced. Last but not least, QDRL can complete the task in a shorter time than LyCD, regardless of the minimum tolerated accuracy. This is an interesting finding, and it proves that QDRL can provide a reference for scenarios that require high accuracy.

*Performance Comparison Under Different Number of Nodes:* There are more and more nodes in IIoT, and whether the proposed scheme can adapt to this change will become an important metric of whether the scheme can be accepted. As depicted in Fig. 12, we evaluate the performance comparison

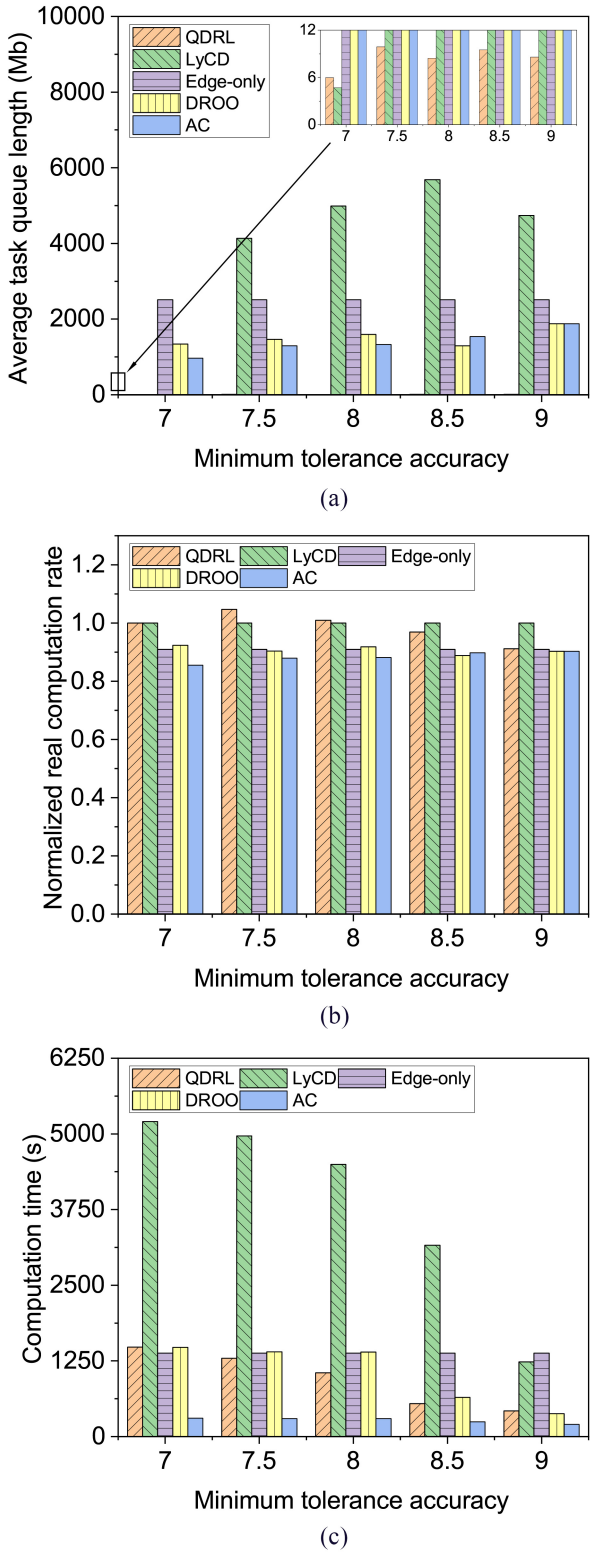



Fig. 11. Performance comparison under different minimum tolerance accuracy. (a) Comparison of the average task queue length. (b) Comparison of the normalized RCR. (c) Comparison of the computation time.

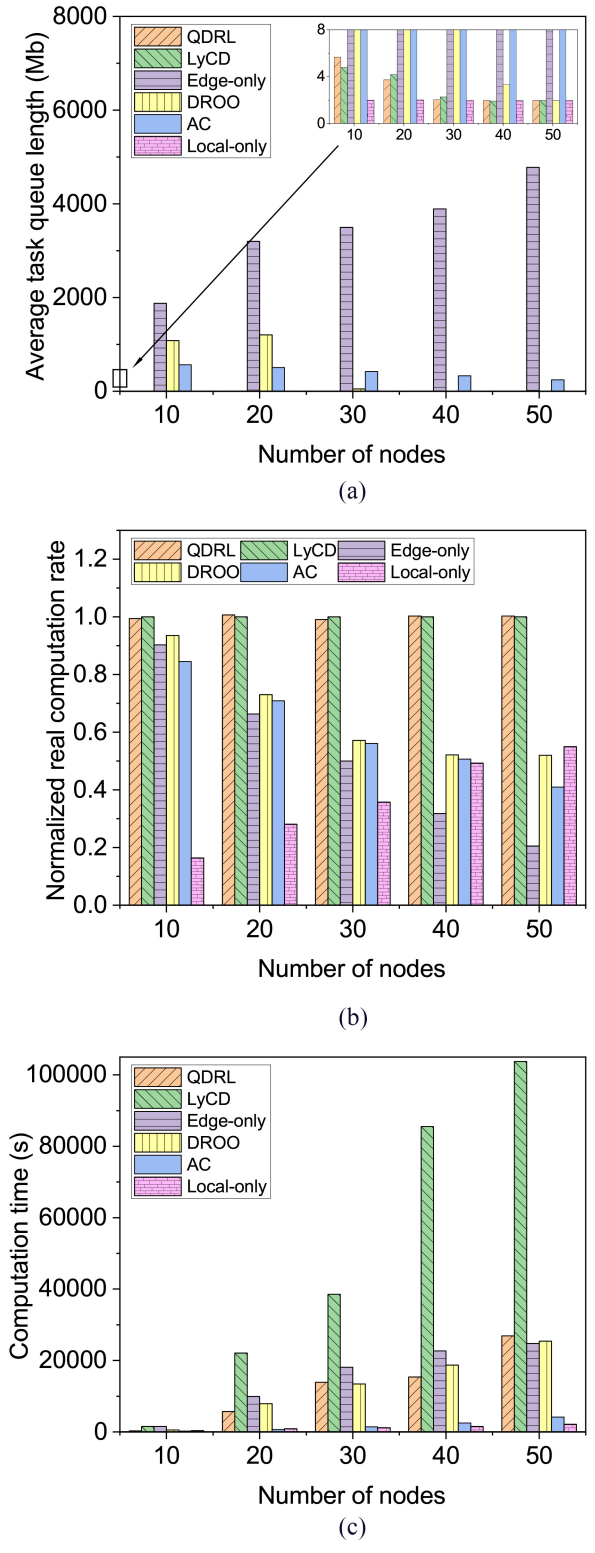



Fig. 12. Performance comparison under different number of nodes. (a) Comparison of the average task queue length. (b) Comparison of the normalized RCR. (c) Comparison of the computation time.

of different schemes under varying numbers of nodes. Thanks to the reasonable scheduling of the task queue and the edge–end collaboration in QDRL, no matter how the node number changes, the average task queue length and RCR of QDRL will not change too much, which proves the advantage of QDRL. Although the computation time of QDRL increases with the number of nodes, it always keeps the computation time low similar to most of the comparison schemes. This provides an opportunity for the real deployment of QDRL.

## VI. Conclusion

Although the IIoT has great application value in industrial environmental monitoring, sensors in the above scenarios usually face severe energy and computing capability bottlenecks and cannot achieve long-term operation. To address the above problems, previous have made great efforts. However, they ignore the edge–end collaboration or ignore the impact of the task queue backlog, resulting in low system revenue. To this end, we design a queue-aware computation offloading scheme based on the AC framework, namely, QDRL. First, the long-term operation of the system is represented as an M-SMIP problem, which is converted into a per-frame deterministic problem by the Lyapunov optimization technique. The above problem contains two subproblems: resource allocation and computation offloading, which are strongly coupled and difficult to solve directly. Subsequently, we design a queue-aware computation offloading scheme based on AC to address these subproblems separately. The Actor module is implemented based on a deep learning model and quantization strategy COGA for generating computation offloading actions. Integrating mathematical reasoning and learning-based methods into a Critic module for resource allocation. Finally, extensive experimental results show that QDRL has advantages in average task queue length and normalized RCR compared with four baseline algorithms, approaching or even surpassing the approximate optimal algorithm. In addition, QDRL still maintains good performance in terms of computation time, which shows that QDRL is suitable for real implementation.

In the future, there is potential to expand the proposed model to support cloud–edge–device collaboration architecture to further improve system revenue. Also, the proposed algorithm may be integrated with the graph neural network to handle tasks with spatial structure.

## References


[1] L. Ren, Z. Jia, Y. Laili, and D. Huang, "Deep learning for time-series prediction in IIoT: Progress, challenges, and prospects," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jul. 11, 2023, doi: 10.1109/TNNLS.2023.3291371.

[2] W. Mao, Z. Zhao, Z. Chang, G. Min, and W. Gao, "Energy-efficient industrial Internet of Things: Overview and open issues," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7225–7237, Nov. 2021.

[3] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multiuser Industrial Internet of Things through edge computing," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 2954–2966, Feb. 2023.

[4] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, "Toward edge-based deep learning in Industrial Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4329–4341, May 2020.

[5] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[6] S. Bi and R. Zhang, "Distributed charging control in broadband wireless power transfer networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3380–3393, Dec. 2016.

[7] P. Wei et al., "Reinforcement learning-empowered mobile edge computing for 6G edge intelligence," *IEEE Access*, vol. 10, pp. 65156–65192, 2022.

[8] H. Zhou, T. Wu, X. Chen, S. He, D. Guo, and J. Wu, "Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 6144–6159, Oct. 2023.

[9] R. Fantacci and B. Picano, "A matching game with discard policy for virtual machines placement in hybrid cloud-edge architecture for industrial IoT systems," *IEEE Trans. Ind. Informat.*, vol. 16, no. 11, pp. 7046–7055, Nov. 2020.

[10] C. Tang et al., "A mobile cloud based scheduling strategy for Industrial Internet of Things," *IEEE Access*, vol. 6, pp. 7262–7275, 2018.

[11] Z. Zhou, M. Shojafar, J. Abawajy, H. Yin, and H. Lu, "ECMS: An edge intelligent energy efficient model in mobile edge computing," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 1, pp. 238–247, Mar. 2022.

[12] K. Li, "Design and analysis of heuristic algorithms for energy-constrained task scheduling with device-edge-cloud fusion," *IEEE Trans. Sustain. Comput.*, vol. 8, no. 2, pp. 208–221, Apr.–Jun. 2023.

[13] X. Deng, J. Zhang, H. Zhang, and P. Jiang, "Deep-reinforcement-learning-based resource allocation for cloud gaming via edge computing," *IEEE Internet Things J.*, vol. 10, no. 6, pp. 5364–5377, Mar. 2023.

[14] B. Yang, X. Cao, X. Li, Q. Zhang, and L. Qian, "Mobile-edge-computing-based hierarchical machine learning tasks distribution for IIoT," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2169–2180, Mar. 2020.

[15] W. Fan, S. Li, J. Liu, Y. Su, F. Wu, and Y. Liu, "Joint task offloading and resource allocation for accuracy-aware machine-learning-based IIoT applications," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3305–3321, Feb. 2023.

[16] H. Materwala, L. Ismail, R. M. Shubair, and R. Buyya, "Energy-SLA-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks," *Future Gener. Comput. Syst.*, vol. 135, pp. 205–222, Oct. 2022.

[17] T. Pamuklu, A. C. Nguyen, A. Syed, W. S. Kennedy, and M. Erol-Kantarci, "IoT-aerial base station task offloading with risk-sensitive reinforcement learning for smart agriculture," *IEEE Trans. Green Commun. Netw.*, vol. 7, no. 1, pp. 171–182, Mar. 2023.

[18] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synth. Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, 2010.

[19] I. T. Christou, N. Kefalakis, J. K. Soldatos, and A.-M. Despotopoulou, "End-to-end industrial IoT platform for quality 4.0 applications," *Comput. Ind.*, vol. 137, May 2022, Art. no. 103591.

[20] M. Al-Amin et al., "Fusing and refining convolutional neural network models for assembly action recognition in smart manufacturing," *Proc. Inst. Mech. Eng. C, J. Mech. Eng. Sci.*, vol. 236, no. 4, pp. 2046–2059, 2022.

[21] R. Usamentiaga, D. G. Lema, O. D. Pedrayes, and D. F. Garcia, "Automated surface defect detection in metals: A comparative review of object detection and semantic segmentation using deep learning," *IEEE Trans. Ind. Appl.*, vol. 58, no. 3, pp. 4203–4213, May/Jun. 2022.

[22] T. Shen et al., "SOTER: Guarding black-box inference for general neural networks at the edge," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2022, pp. 723–738.

[23] L. Zeng, P. Huang, K. Luo, X. Zhang, Z. Zhou, and X. Chen, "Fograph: Enabling real-time deep graph inference with fog computing," in *Proc. ACM Web Conf.*, 2022, pp. 1774–1784.

[24] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Comput. Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[25] C. Fang et al., "DRL-driven joint task offloading and resource allocation for energy-efficient content delivery in cloud-edge cooperation networks," *IEEE Trans. Veh. Technol.*, early access, Jul. 24, 2023, doi: 10.1109/TVT.2023.3297362.

[26] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[27] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, "Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4988–4998, Jul. 2020.

[28] Z. Zhang, H. Chen, M. Hua, C. Li, Y. Huang, and L. Yang, "Double coded caching in ultra dense networks: Caching and multicast scheduling via deep reinforcement learning," *IEEE Trans. Commun.*, vol. 68, no. 2, pp. 1071–1086, Feb. 2020.

[29] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[30] W. Zhang et al., "Deep reinforcement learning based resource management for DNN inference in industrial IoT," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 7605–7618, Aug. 2021.

[31] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[32] R. Zhang and C. K. Ho, "MIMO broadcasting for simultaneous wireless information and power transfer," *IEEE Trans. Wireless Commun.*, vol. 12, no. 5, pp. 1989–2001, May 2013.
[33] K. Zheng, X. Liu, B. Wang, H. Zheng, K. Chi, and Y. Yao, "Throughput maximization of wireless-powered communication networks: An energy threshold approach," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1292–1306, Feb. 2021.
[34] Q. Hong, F. Liu, D. Li, J. Liu, L. Tian, and Y. Shan, "Dynamic sparse R-CNN," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 4723–4732.
[35] Y. Hua, M. Sevegnani, D. Yi, A. Birnie, and S. Mcaslan, "Fine-grained RNN with transfer learning for energy consumption estimation on EVS," *IEEE Trans. Ind. Informat.*, vol. 18, no. 11, pp. 8182–8190, Nov. 2022.
[36] Y. Jiang, J. Liu, D. Xu, and D. P. Mandic, "UAdam: Unified adam-type algorithmic framework for non-convex stochastic optimization," 2023, *arXiv:2305.05675*.
[37] J. Xu, B. Yang, Y. Liu, C. Chen, and X. Guan, "Joint task offloading and resource allocation for multihop Industrial Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 21, pp. 22022–22033, Nov. 2022.
[38] P. Wang et al., "Scaled ReLu matters for training vision transformers," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, 2022, pp. 2495–2503.
[39] Z. Pan, Z. Gu, X. Jiang, G. Zhu, and D. Ma, "A modular approximation methodology for efficient fixed-point hardware implementation of the sigmoid function," *IEEE Trans. Ind. Electron.*, vol. 69, no. 10, pp. 10694–10703, Oct. 2022.
[40] G. Tao et al., "Better trigger inversion optimization in backdoor scanning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 13368–13378.
[41] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021.
[42] M. J. Schuetz, J. K. Brubaker, and H. G. Katzgraber, "Combinatorial optimization with physics-inspired graph neural networks," *Nat. Mach. Intell.*, vol. 4, no. 4, pp. 367–377, 2022.
[43] Y. Zhang et al., "Free lunch for domain adversarial training: Environment label smoothing," 2023, *arXiv:2302.00194*.
[44] "QDRL." 2023. [Online]. Available: https://github.com/xuaikun/QDRL.git

**Aikun Xu** received the M.S. degree from Central South University, Changsha, China, in 2022, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

His research interests include deep learning, graph neural network, deep reinforcement learning, scheduling, electric vehicles, and edge computing.

**Zhigang Hu** received the B.S., M.S., and Ph.D. degrees from Central South University (CSU), Changsha, China, in 1985, 1988, and 2002, respectively.

In 2002, he joined CSU, where he is a Professor with the School of Computer Science and Engineering. He has published over 200 research papers. His research interests include radar signal processing and classification/recognition, high-performance computing, and cloud computing.

**Xinyu Zhang** received the master's degree from Central South University, Changsha, China, in 2017, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

His main research interests include edge computing, cloud computing, and federated deep reinforcement learning.

**Hui Xiao** received the B.E. degree from Shandong University, Jinan, China, in 2017, and the M.E. degree from Central South University, Changsha, China, in 2020, where she is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering.

Her main research interests are in the area of mobile-edge computing and cloud computing.

**Hao Zheng** (Graduate Student Member, IEEE) received the M.S. degree from Central South University, Changsha, China, in 2021, where he is currently pursuing the Ph.D. degree in the research group of Zhigang Hu with the School of Computer Science and Engineering.

His research interests include synthetic aperture radar image processing, transfer learning, and computer vision.

**Bolei Chen** received the B.S. degree from the School of Computer Science and Engineering, South Central University for Nationalities, Wuhan, China, in 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Central South University, Changsha, China.

His research interests include deep reinforcement learning, robotic navigation, and autonomous exploration.

**Meiguang Zheng** received the B.S. and Ph.D. degrees in computer science from Central South University, Changsha, China, in 2005 and 2011, respectively.

She is currently an Associate Professor with the School of Computer Science and Engineering, Central South University. She is currently leading some research projects supported by the National Natural Science Foundation of China. Her research interests include federated learning, distributed machine learning, computer vision, and edge computing.

**Ping Zhong** (Member, IEEE) received the Ph.D. degree in communication engineering from Xiamen University, Xiamen, China, in 2011.

She is currently an Associate Professor with the School of Computer Science and Engineering, Central South University, Changsha, China. Her research interests include machine learning, data mining, and network protocol design.

Dr. Zhong is a member of ACM, CCF, and IEICE.

**Yilin Kang** received the Ph.D. degree in computer science from Nanyang Technological University, Singapore.

She is currently an Assistant Professor with the School of Computer Science, South Central University for Nationalities (SCUN), Wuhan, China. Prior to joining SCUN, she was a Research Fellow with the NTU-UBC Joint Research Centre of Excellence in Active Living for the Elderly (LILY), Singapore. Her current research interests include human-centric computing, brain-inspired intelligent agent, cognitive and neural systems, and human–computer interaction.

Dr. Kang serves as a PC Member of AAAI 2020, AAAI 2019, and IJCAI 2016 and an OC Member of IEEE WI/IAT 2015. She serves as a reviewer for several major journals, such as IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, and *Journal of Autonomous Agents and Multi-Agent Systems*.

**Keqin Li** (Fellow, IEEE) is currently a SUNY Distinguished Professor of Computer Science with the State University of New York at New Paltz, New Paltz, NY, USA. He has published over 620 journal articles, book chapters, and refereed conference papers. His current research interests include cloud computing, fog computing and mobile-edge computing, energy-efficient computing and communication, embedded systems, cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, and intelligent and soft computing.

Prof. Li received several best paper awards. He currently serves or has served on the editorial boards of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, and IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.