NOMA-Enabled Cooperative Computation Offloading for Blockchain-Empowered Internet of Things: A Learning Approach

Zhenni Li, *Member, IEEE*, Minrui Xu, Jiangtian Nie, *Student Member, IEEE*, Jiawen Kang, Wuhui Chen, *Member, IEEE*, and Shengli Xie, *Fellow, IEEE*

***Abstract*—Blockchain technologies allow the Internet of Things (IoT) to build trust among various interest parties. For the resource-limited IoT devices, offloading computation-intensive tasks (blockchain verification and mining tasks, and data process tasks) to edge servers for execution is considered as a promising solution in mobile-edge computing. However, conventional methods (such as linear programming or game theory) for the computation offloading problem cannot achieve long-term performance while the existing deep reinforcement learning (DRL)-based algorithms suffer from slow convergence, lack of robustness, and unstable performance. In this article, we propose a multiagent DRL framework to achieve long-term performance for cooperative computation offloading, in which a scatter network is adopted to improve its stability and league learning is introduced for agents to explore the environment collaboratively for fast convergence and robustness. First, we study the nonorthogonal multiple access-enabled cooperative computation offloading problem and formulate the joint problem as a Markov decision process by considering both the blockchain mining tasks and data processing tasks. Second, to avoid useless exploration and unstable performance, we initially train an intelligent agent represented by scatter networks using conventional expert strategies. Third, in order to enhance the performance, we subsequently establish a hierarchical league where agents collaborate with others to explore the environment. Finally, our experimental results demonstrate that our algorithm could perform better in terms of reducing energy cost and delay cost, and shortening almost 60% of the training time compared with the state-of-the-art approaches.**

***Index Terms*—Blockchain, cooperative computation offloading, deep reinforcement learning (DRL), edge computing.**

I. Introduction

BLOCKCHAIN is an emerging decentralized and autonomous technology that can assist the Internet of Things (IoT) to build trust among multiple parties with conflicting interests [1]–[4]. However, the consensus of blockchain requires lots of computing resources, leading to IoT devices' lack of computing resources. To alleviate the tension between resource-hungry mining tasks and resource-constrained IoT devices, mobile-edge computing (MEC) has been proposed as an enabler, which can provide the computation resource to IoT devices with low latency [5]. MEC is to deploy the cloud-computing capabilities to the edge of a pervasive radio network, so that IoT devices can offload the computation via nonorthogonal multiple access (NOMA) to the edge servers for capability augmentation [6], [7]. With the objective of enabling multiple users to share the same time–frequency resource, NOMA can enhance MEC to serve a large number of users by supporting massive connectivity. However, to make the offloading decision in NOMA-enabled multihop cooperative computing environments is a challenging issue.

To solve the computation offloading problem, conventional methods, such as linear programming and game theory are widely investigated [8]. For example, the multihop cooperative computation offloading optimization problem was formulated as a game for robot swarms [9]. However, there are some drawbacks to these conventional methods. First, they often presuppose that the system environment is determined, but in reality, it is a turntable in dynamic balance. Second, the solutions based on conventional methods ignore the essential that the perception of devices in the system is dynamic, and moreover, the interaction between devices and the environment is nondeterminism. Third, conventional methods are often myopic that they seek the optimal solutions for one-slot systems, however, a long-term return optimization is what the systems really demanded.

Manuscript received March 29, 2020; revised July 4, 2020; accepted August 9, 2020. Date of publication August 14, 2020; date of current version February 4, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61803096, Grant 61722304, Grant 61703113, and Grant 61727810; in part by the Science and Technology Plan Project of Guangzhou under Grant 202002030289; in part by the Key Areas of Research and Development Plan Project of Guangdong under Grant 2019B010147001; and in part by the Major Research Project on Industry Technology of Guangzhou under Grant 201902020014. *(Corresponding author: Shengli Xie.)*

Zhenni Li is with the School of Automation and Guangdong Key Laboratory of IoT Information Technology, Guangdong University of Technology, Guangzhou 510006, China (e-mail: lizhenni2012@gmail.com).

Minrui Xu and Wuhui Chen are with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China (e-mail: xumr3@mail2.sysu.edu.cn; chenwuhui21@gmail.com).

Jiangtian Nie and Jiawen Kang are with the Energy Research Institute @ NTU, Nanyang Technological University, Singapore (e-mail: jnie001@e.ntu.edu.sg; kavinkang@ntu.edu.sg).

Shengli Xie is with the Guangdong–HongKong–Macao Joint Laboratory for Smart Discrete Manufacturing and 111 Center for Intelligent Batch Manufacturing Based on IoT Technology, Guangdong University of Technology, Guangzhou 510006, China (e-mail: shlxie@gdut.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3016644

To overcome these challenges of the conventional methods, deep reinforcement learning (DRL) is developed [10]. DRL uses the Markov decision process (MDP) framework to define the interaction between the intelligent agent and environment in terms of states, actions, and rewards [11], where the intelligent agent is a parameterized approximator that addresses the classical "curse of dimensionality" in the control theory. For example, a continuous action space-based DRL approach named deep deterministic policy gradient (DDPG) is proposed by Chen and Wang [12] to learn efficient computation offloading policies independently at each mobile user. However, there are still some urgent issues [13] that need to be paid attention to.

1) The existing DRL-based algorithms take a long clock-runtime to achieve convergence because they initialize their agents with the random initialization, which leads to lots of useless exploration of agents in the initial phase of training. It reduces the availability of DRL-based methods in practical applications since long training time makes them unable to adjust to the changes in the system in time.
2) The existing DRL-based works to address the MEC problems are often based on the single-agent learning algorithm. The final performances of their algorithms are affected greatly by different superparameters such as random seed, which is shown in the agent converges in the local optimal solution.
3) The performances of the existing DRL-based algorithms fluctuate sharply with the dynamic changes of the environment. During their construction of intelligent agents' neural networks, a tremendous difference in the numerical value and meaning between the computation states and communication states will affect the performances of the existing DRL-based algorithm.

To address these issues, we propose a multiagent DRL framework to minimize the long-term cost of cooperative computation offloading, where the scatter network is adopted to improve its stability, and moreover, league learning is introduced for agents to explore the environment collaboratively for fast convergence and robustness. To address the first issue, we devote our effort to balance exploration and exploitation. By using the existing expert strategies, an intelligent agent avoids useless exploration arising from random initialization at the initial phase of training. We initially train our intelligent agent using the strategies designed for the OMA-enabled system. To address the second issue, our intelligent agent is empowered with a multiagent framework called league learning, which allows hierarchical agents differing from their desires to explore to interact with the environment collaboratively. By training on the experiences from the league, the intelligent agent can discover more novel strategies and has robust under different super parameters. To address the third issue, in order to achieve stable performance in spite of the dynamics in the environment, we construct scatter networks for an intelligent agent to distinguish information observed from the environment and actions.

The main contributions of this article can be summarized as follows.

1) We design an efficient multiagent DRL algorithm, which allows hierarchical agents within a league to explore the environment collaboratively and enhances the robustness of our algorithm.
2) We propose an expert strategy-based initialization method to avoid useless exploration at the initial phase of the intelligent agent's training, which greatly reduces the convergent time.
3) To achieve stable performance in spite of the dynamics of the environment, we construct scatter networks for the intelligent agent to distinguish different observations and actions.
4) Simulation results demonstrate the proposed algorithm has a stable performance and reduces almost 60% of training time compared with state-of-the-art approaches.

The remainder of this article is outlined as follows. We discuss the related work in Section II. We present the system model in Section III. We propose the multiagent reinforcement learning framework in Section IV. We simulate our result in Section V. Finally, we conclude this article in Section VI.

II. Related Work

A. MEC for Blockchain-Empowered IoT

Many works in the rising blockchain-empowered IoT field have been proposed to tackle challenges in systems, models, and applications [14]. For example, Xiong *et al.* [15] have proposed a lightweight infrastructure of the proof-of-work-backed blockchains using a game-theoretic approach. Furthermore, a general architecture combining blockchain and IoT systems has been presented by the work [16]. A multileader–multifollower game-theoretic approach has been proposed by Xiong *et al.* [17] that investigates the optimum solution in interactions among the cloud/edge providers and miners in the blockchain. Zhou *et al.* [18] have attempted to cover the existing scaling solutions for blockchain and classify them by level.

B. Conventional Methods for MEC

Extensive works focus on effective methods to improve performance in computation offloading problems [19]. A free-bound mechanism and a multihop cooperative messaging mechanism have been studied by Hong *et al.* [20], by formulating the joint problem as a potential game in which devices determine their equilibrium. Ning *et al.* [21] have proposed a hybrid computation offloading framework for real-time traffic management, aiming at maximizing the link rate by enabling various offloading schemes that empowering by the integration of NOMA and MEC technology. A potential game whose devices can make their decisions autonomously has been proposed by Chen *et al.* [22] to minimize the economic cost of blockchain-empowered devices when they cannot directly connect to the edge servers or cloud data centers. However, the conventional algorithms ignored the dynamics and randomness of the wireless environment and their solutions are designed for the short term.

C. DRL-Based Algorithms for MEC

With the advent of DRL [23], the above issues can be addressed by formulating the computation offloading problems as the Markov decision processes and learning the optimal policies by interacting with the environments [24]. Many existing works have studied the multiuser computation offloading using single-agent DRL algorithms, such as Chen *et al.* [25] have been motivated by the additive structure of the utility function. They have proposed a Q-function decomposition technique is combined with the double deep Q-learning (DQN), which leads to a novel learning algorithm for the solving of stochastic computation offloading. A dueling DQN approach has been studied by Qiu *et al.* [13] to address the joint optimization problem consists of agent mining and cloud mining in blockchain-enabled IoT. However, the above DRL-based approaches suffer from slow convergence caused by high-dimensional action space. In order to improve the slow convergence problems in DQN, Feng *et al.* [26] have developed an A3C-based cooperation computation offloading and resource allocation to maximize the computation rate of MEC systems and the transaction throughput of blockchain systems. To speed up the convergence, Qiu *et al.* [1] have adopted an adaptive genetic algorithm [27] into the exploration of DRL that effectively avoids useless exploration and speeds up the convergence without decreasing the performance. Asheralieva and Niyato [28] have modeled a stochastic Stackelberg game under private information. They have designed a hierarchical learning framework for this game to converge to the stable states, in which miners' actions are the best responses to the optimal price assigned by the service provides. However, using single-agent DRL algorithms to address the computation offloading problems in the multiuser case suffers from slow convergence and unstable performance. That is because of the nonstationarity of the multiuser environment that users constantly change their strategies during the training phase.

Fortunately, the recent breakthrough of multiagent DRL provides alternative approaches [29] to address the nonstationarity of the multiuser environment by balancing exploration and exploitation. Zhan *et al.* [30] have applied the game theory to the algorithm design and formulated the problem as a partially observable MDP to solve the edge computing problem when users refuse to expose their own information. Zhan *et al.* [31] have investigated multiagent DRL-based techniques to address the problem of assigning a satisfactory but profitable amount of incentives to multileader and multifollower as a mobile crowdsensing game. However, most studies have not taken into account multiple agents explore the environment collaboratively. In this article, we propose a multiagent DRL framework where multiple agents exploring the environment collaboratively for fast convergence and robustness while the scatter network is adopted to guarantee its stability.

III. System Model and Problem Formulation

The notations used in this paper are summarized in Table I. We first introduce our system model. As shown in Fig. 1, we consider a uplink NOMA-based cooperative computation offloading for blockchain-empowered IoT, which consists of IoT devices and edge servers. We denote the set of IoT devices as $\mathcal{N} = \{1, \ldots, n, \ldots, N\}$ and the set of edge servers as $\mathcal{B} = \{1, \ldots, b, \ldots, B\}$.

TABLE I
Notations Used in This Article

Symbol	Definition
$\mathcal{N}$	Set of IoT devices
$\mathcal{B}$	Set of edge servers
T	Length of episodes
$g_{n,l}$	Channel gain between n and relay l
p_n	Transmit power of IoT device n
$r^{co}_{n,b}$	Cooperative transmission rate achieved by n
$L(n)$	Cluster of relays assist IoT device n
W	Channel bandwidth in the system
$\mathcal{T}_n$	Hybrid task of n
s_n	Data size of n
c_n	CPU cycles per data size of n
e_n	Consumed energy of n
d_n	Consumed delay of n
a_n	Decision profile of device n
h_n, h_b	Consumed energy per CPU cycle of device n andserver b
f_n, f_b	Computation capacity of device n, edge server b
T^{sw}_l	Switch delay of relay l
E^{sw}_l	Switch energy of relay l
$\mathcal{O}_{\text{total}}$	Total overhead in the system
α^e_n, α^d_n	Weight of energy and Weight of delay of device n
$x(t)$	Joint state at decision epoch t
$u(t)$	Joint action at decision epoch t
$C(t)$	Expected future sum cost from t
π	Agent's policy
Q	Agent's action-value function
γ	Discounting factor
$\theta^{\pi}, \theta^{\pi'}$	Parameters of policy network and target policy network
$\theta^{Q}, \theta^{Q'}$	Parameters of critic network and target critic network
α, β	Learning rate of policy network and critic network
$T_{\text{MEA}}, T_{\text{LEA}}$	Reset period of MEA and LEA
$\tau_{\text{MEA}}, \tau_{\text{LEA}}$	Reset ratio of MEA and LEA

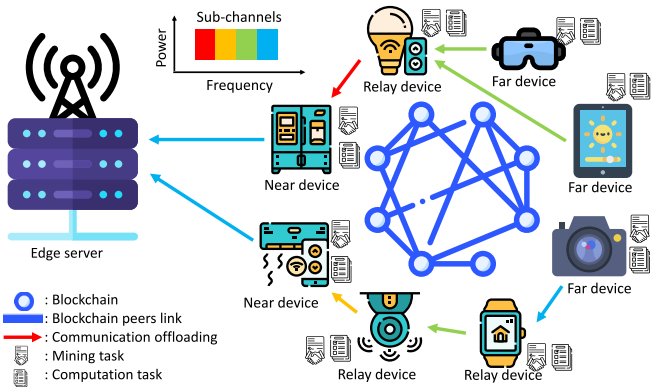

Fig. 1. Illustration of the NOMA-enabled cooperative computation offloading framework for the blockchain-empowered IoT.

A. Communication Model

We first introduce the communication model for NOMA in MEC. Here, the system consists of an edge server and N IoT devices, and each IoT device has the ability to transmit

and receive data. Different from the conventional orthogonal frequency-division multiple access (OFDMA), the same subchannel in NOMA can be shared in multiple users to gain multiplexing benefits. Multi-input–multi-output (MIMO) [32] technology enables IoT devices to assist other devices offloading computation to the edge servers, i.e., cooperative computation offloading [20]. As illustrated in Fig. 1, the edge servers can provide resource-rich computation capabilities for IoT devices, and devices can offload their computation tasks to the edge servers by making full use of the subchannel.

1) Direct Transmission Phase: In the direct transmission phase, IoT devices can only transmit their hybrid tasks to edge servers directly. The received signal y_b at the edge server b is given by

$$y_b = \sum_{n\in\mathcal{N}} g_n p_n x_n + v_b \tag{1}$$

where $g_{n,b}$ denotes the Rayleigh fading channel coefficient from the device n to the edge server b and x_n is the message of device n. v_b denotes the additive Gaussian noise at the edge server b. The use of NOMA implies $\sum_{n\in\mathcal{N}} p_n^2 = 1$, where p_n is the power allocation coefficient for the IoT device n.

Without loss of generality, assume that the IoT devices are ordered based on their channel quality, i.e.,

$$|g_1|^2/\delta_1^2 < |g_2|^2/\delta_2^2 < \cdots < |g_{n-1}|^2/\delta_{n-1}^2 < |g_n|^2/\delta_n^2. \tag{2}$$

Successive interference cancelation (SIC) is a decoding method in NOMA, where the users with strong channel conditions are capable of removing the interference of weak users. In uplink NOMA, the signal first decoded at the edge server will experience interference from all users in the cluster with relatively weaker channels. That is, the transmission of the user with the highest channel gain experiences interference from all users within its cluster, whereas the transmission of the user with the lowest channel gain receives zero interference from the users in its cluster. With the use of SIC, the signal-to-interference-plus-noise ratio (SINR) [21] from the IoT device n to edge server b is given by

$$\mathrm{SINR}_{n,b} = \frac{g_n p_n}{\sum_{k=1}^{n-1} g_k p_k + \delta_b^2} \tag{3}$$

where δ_b^2 denotes the variance of the additive white Gaussian noise on the receiver b. However, it is weak when the user is far away from the edge server.

2) Cooperative Transmission Phase: During this phase, the IoT devices cooperate with each other using a short-range communication channel. We denote that $L(n)$ is the cluster of relays assisting the IoT device n offloading its tasks to the edge server. The relay devices within the cluster are denoted by $l \in L(n)$. With the help of relay devices, the edge server b can observe the superposition as

$$y_{n,b} = \sum_{n\in\mathcal{N}} \sum_{l\in L(n)} g_l p_l x_{n,l} + v_b \tag{4}$$

where v_b denotes the noise at edge server b. We then compute the cooperative $\mathrm{SINR}_{n,b}^{co}$ [26] by adding the source SINR and the truncated cooperative SINR along the relays as

$$\mathrm{SINR}_{n,b}^{co} = \mathrm{SINR}_{n,b} + \sum_{l\in L(n)} \min\left(\mathrm{SINR}_{l,b}, \mathrm{SINR}_{l-1,l}\right) \tag{5}$$

where $(l-1, l)$ denotes the pair of relays in cooperative transmission. Comparing (3) with (5), cooperative transmission conspicuously increases the SINR received at edge servers and boosts reception reliability. In our considered dense networking scenario, according to the definition of $\mathrm{SINR}_{n,b}^{co}$ and the Shannon theory [33], the cooperatively achievable transmit rate can be written as

$$r_{n,b}^{co} = W \cdot \log\left(1 + \mathrm{SINR}_{n,b}^{co}\right) \tag{6}$$

where W is the allocated channel bandwidth in the NCoCO system. Moreover, this would positively affect the performance of MEC if well controlling the interference among the cooperative system.

B. Computation Model

We then discuss the computation model. Here, every IoT devices in blockchain-empowered IoT generates the hybrid tasks that consist of mining tasks and computation tasks each decision slot, and the IoT device n's task $\mathcal{T}_n = (s_n, c_n)$ is characterized by two parameters, i.e., s_n denotes the size of the task and c_n denotes the required CPU cycles per data size. The tasks arrive at IoT devices following a Poisson process. The IoT devices within the coverage of edge servers can directly offload their hybrid tasks to one of the edge servers via wireless access. The other IoT devices, offloading their tasks to edge servers via relay devices, are not be covered by the edge servers. We slice the time in the blockchain-empowered IoT system into the decision slot $t \in \{1, \ldots, T\}$ [34], where a decision slot is a much slower time scale than that of tasks arrival and offloading. In NCoCO for blockchain-empowered IoT, energy cost and delay cost are the main considerations in system optimization, so our computation model consists of an energy model and a delay model.

1) Energy Model: In NCoCO, the energy cost is divided into three parts: the first one is the local computing energy consumption of IoT devices, the second one is the computing energy consumption of edge servers to accomplish tasks, and the last one is the transmit energy consumption during data transmission. Denoting h_n as the consumed energy per CPU cycle of the IoT device n, it can be obtained by the measurement method in [35]. The n device's consumed energy $e_n^l(t)$ in the local processing at decision slot t is given as

$$e_n^l(t) = s_n(t) c_n(t) h_n(t). \tag{7}$$

Edge servers are much more resource-rich than a single IoT device, which prefer offloading their tasks to edge servers for execution. We can then compute the offload processing consumed energy as

$$e_n^o(t) = s_n(t) c_n(t) h_b(t) + p_n(t) \frac{s_n(t)}{r_{n,b}^{co}(t)} + \sum_{l\in L(n)} T_l^{sw}(t) \tag{8}$$

where h_b is the consumed energy per CPU cycles at the edge server b. For the IoT device n, its consumed energy at decision

slot t needs to be determined according to the decision profile $a_n(t) \in \{0, 1\}$, and $p_n(t)$ is the transmit power that the device n is allocated according to its remained energy. For the energy cost e_n for the device n, we have that

$$e_n(t) = a_n(t)e_n^l(t) + a_n(t)e_n^o(t) \tag{9}$$

where s_n is the transmit size of n's tasks, and when $a_n(t) = 1$, the hybrid tasks of the IoT device n are processed locally. Otherwise, the hybrid tasks are offloaded to the edge servers for execution.

2) Delay Model: We can then compute the local processing consumed delay for the IoT device n as

$$d_n^l(t) = \frac{s_n(t)c_n(t)}{f_n(t)} \tag{10}$$

where s_n is the task size, c_n is the required CPU cycles per data size of the device n, and f_n is the CPU frequency of the device n.

The offload processing consumed delay for the IoT device n is calculated as

$$d_n^o(t) = \frac{c_n(t)s_n(t)}{f_b(t)} + \frac{s_n(t)}{r_{n,b}^{co}(t)} + \sum_{l \in L(n)} T_l^{sw}(t) \tag{11}$$

where f_b is the computation capacity of the edge server b, and $r_{n,b}^{co}$ is the transmit rate achieved by cooperative NOMA. In the cooperative offloading phase, it cannot be ignored by the switch delay T^{sw} for each relay $l \in L(n)$.

So the delay for the IoT device n can be formulated as

$$d_n(t) = a_n(t)d_n^l(t) + a_n(t)d_n^o(t) \tag{12}$$

where a_n is the decision profile for the device n deciding locally executes its tasks or offloads them to the edge servers for execution.

3) Overhead: Following (9) and (12), the overhead of the blockchain-empowered IoT system can be expressed as

$$\mathcal{O}_{total}(t) = \sum_{n \in \mathcal{N}} \left(\alpha_n^e e_n(t) + \alpha_n^d d_n(t) \right) \tag{13}$$

where $\alpha_n^e = \beta_n^e / e_n^l$ and $\alpha_n^d = \beta_n^d / d_n^l$ while $0 \le \beta_n^e, \beta_n^d \le 1$. We set $\beta_n^d > \beta_n^e$ to intensify the latency aspect when the device is executing some latency-sensitive tasks. Oppositely, we can set $\beta_n^e > \beta_n^d$ that makes the devices more susceptive to energy consumption, if the battery capacity of the device is petite.

C. MDP for NMCO

In the following section, we put forward the observation space, action space, state transition probability, and cost function for the MDP in the NOMA-enabled cooperative computation offloading system for blockchain-empowered IoT. In the control theory, the MDP problem is defined by a tuple $< \mathcal{X}, \mathcal{U}, \mathcal{P}, \mathcal{C} >$, where $\mathcal{X}$ is the state space of the system, $\mathcal{U}$ is the action space of the system, $\mathcal{P}$ is the state transition probabilities, and $\mathcal{C}$ is the cost space.

1) State Space: We define the joint state space at the current decision slot t as $x(t) = [x_{\text{cmp}}(t), x_{\text{com}}(t)]$, which consists of the computation state space $x_{\text{cmp}}(t)$ and the communication state space $x_{\text{com}}(t)$.

Computation State: We denote the computation state by $x_{\text{cmp}}(t)$, as shown in the following:

$$x_{\text{cmp}}(t) \triangleq [\text{ID}, \mathbf{T}(t), \mathbf{F}(t), \mathbf{E}(t), \mathbf{R}(t-1)] \tag{14}$$

where *ID* is the IoT devices' identity number. $s_n(t)$ and $c_n(t)$ in $\mathbf{T}(t) = [s_1(t), c_1(t), s_2(t), c_2(t), \ldots, s_n(t), c_n(t)]$ denote the task size and the required CPU cycles per data size of the device $n \in \mathcal{N}$, respectively. $\mathbf{F}(t) = [f_1(t), f_2(t), \ldots, f_n(t), \ldots, f_b(t)]$ is the computation capacities of devices and edge servers, and $\mathbf{R}(t-1) = [r_1(t-1), r_2(t-1), \ldots, r_n(t-1)]$ is the feedback signal from last decision slot $t-1$ that devices observe from the environment.

Communication State: The signal of the communication state is always smaller than the computation state, and it is witty to process them respectfully. We denote the communication state as

$$x_{\text{com}}(t) \triangleq [\text{ID}, \mathbf{G}(t)] \tag{15}$$

where $\mathbf{G}(t) = [g_1(t), g_2(t), \ldots, g_n(t)]$ denotes the channel gains in the system. In order to make the observations better as the input of the neural network, we need to apply some adjustments to the raw input. We first use min–max normalization to process the computation state $x_{\text{cmp}}(t)$ and the communication state $x_{\text{com}}(t)$, and then use Z-score normalization to process the joint state $x(t)$.

2) Action Space: The joint action space $u(t)$ in the decision slot t, including the offloading decision $\mathbf{a}(t)$, the power allocation decision $\mathbf{P}(t)$, and the constraint offloading distance $\mathbf{L}(t)$, can be defined as $u(t) = [\mathbf{a}(t), \mathbf{P}(t), \mathbf{L}(t)]$. By using the offloading distance, we convert the next-hop choosing problem into a distance decision problem.

Offloading Decision: We denote the joint decision profile as

$$\mathbf{a}(t) = [a_1(t), a_2(t), \ldots, a_N(t)]. \tag{16}$$

Power Allocation Decision: The increase in the transmit energy will amplify the transmit signal and the degree of interference between devices at the same time. However, when the interference is too serious, the transmit rate does not increase or even decrease. Constraining the transmit power is a solution for this tension between transmit interference and transmit signal. We denote the joint transmit power allocation decision by $\mathbf{P}(t)$ as

$$\mathbf{P}(t) = \left[p_1(t), p_2(t), \ldots, p_N(t) \right]. \tag{17}$$

Constraint Distance Decision: The transmit signal fades with the increase of distance, thus affecting the transmit rate. In the NOMA-enabled communication scenario, the increased bandwidth makes the fading more significant. So we want to make the communication efficient by limiting the distance of the next-hop device. The constraint distance $\mathbf{L}(t)$ is shown as follows:

$$\mathbf{L}(t) = [l_1(t), l_2(t), \ldots, l_N(t)]. \tag{18}$$

In cooperative computation offloading, we note that the choice of the next-hop devices will make the communication better. In other words, the distance between the next-hop device to the edge server is closer than that from the source device to the edge server. Moreover, when the distance between the next-hop device and the source device is farther than the distance from the source device to the edge server, the source device is preferred to offload directly to the edge server.

D. State Transition Probability

The probability of a particular state is 0 because the state space is continuous. We assume that the current state and action is $x(t) = x$ and $u(t) = u$; then the probability that the process will leave $x(t)$ and transition into the next state $x(t+1) = x'$ after taking an action $x(t)$ can be expressed as

$$Pr\big(x'|x,u\big) = \int_{\mathcal{R}} f\big(x', c|x, u\big)dc \tag{19}$$

where $f : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \times \mathcal{C} \rightarrow [0, 1]$ is the state transition probability density function mapping current state, next state, current action, and current cost to a transition probability in [0,1].

E. Cost Function

There is a cost space $\mathcal{C} = \{c(x(t), u(t))\}$. We then define the cost function as $c(x(t), u(t)) = \mathcal{O}_{total}(t)$ that $c : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$. In this article, the joint optimization problem is formulated as

$$2\min_{u(t)} \sum_{t=1}^{T} c(x(t), u(t)) \tag{20}$$

$$\text{s.t. } a_n(t) \in \{0, 1\} \tag{21}$$

$$\sum_{n\in\mathcal{N}} p_n(t) \le P_N, p_n(t) \ge 0 \tag{22}$$

$$0 \le l_n(t) \le L_N, \forall n \in \mathcal{N} \tag{23}$$

$$0 \le r^{\text{co}}_{n,b}(t) \le W, \forall n \in \mathcal{N}, \forall b \in \mathcal{B}. \tag{24}$$

Constraint (21) shows each hybrid task can be executed locally or offloaded to edge servers for execution. Constraint (22) indicates the transmit power p_n of the device n should be no more than their limited transmit power P_N. The constraint for offloading distance demonstrated in (23) should be less than the maximum offloading distance L_N. Moreover, through the utilization of SIC, the cooperative transmit rate should never exceed the system bandwidth as demonstrated in (24).

Proposition 1: The NOMA-enabled cooperative computation offloading problem in (20) is NP-hard.

Proof: There are three simplified cases for this problem, i.e., the problem of whether to offload, the joint problem of power allocation, and the routing problem. The problem of whether to offload and routing problem can be reduced to the cardinality bin packing problem, which is NP-hard [19]. In [36], the joint problem of power allocation for NOMA has demonstrated to be NP-hard. Since simplified cases of (20) are NP-hard, Proposition 1 holds. ■

Based on the system model above, in the following section, we will then propose a multiagent DRL framework to learn efficient computation offloading policy by adopting the scatter network and league learning.

IV. Algorithm Design

In this section, we consider the issue of designing a multiagent DRL framework for achieving fast convergence and improving the robustness and stability in performance. Fig. 2 shows that our algorithm maintains a hierarchic league, where upper agents accept the experiences from lower agents for reinforcement learning, and lower agents periodically update their network parameters using that of upper agents and initialization agent. The scatter connection structure of the agent's policy network and critic network are shown in the top of Fig. 2.

A. Agent and Network Design

We design an intelligent agent in the actor–critic and off-policy fashion. It means that the agent maintains a deterministic policy function and an action-value function. They are both parameterized by deep neural networks. The intelligent agent's deterministic policy π can be described as

$$\pi\big(x(t)|\theta^{\pi}\big) = u(t) \tag{25}$$

where θ^{π} is the parameters of the policy network. The policy outputs the action $u(t)$ conditioning on the state $x(t)$. We use the critic network with the parameters θ^{Q} to approximate the action-value function Q^{π} as

$$Q^{\pi}\Big(x, u|\theta^{Q}\Big) = \mathbb{E}_{\pi}[C(t)|x(t), u(t)]. \tag{26}$$

It describes the expected return after taking an action $u(t)$ in the state $x(t)$ and, thereafter, the following cost $C(t)$ is:

$$C(t) = \sum_{i=t}^{T} \gamma^{(i-t)} c(x(i), u(i)). \tag{27}$$

It means the return from a state $x(t)$ as the sum of the discounted future cost with the discounting factor $\gamma \in [0, 1]$. When the target policy is deterministic, we also can make the use of the recursive relationship known as the Bellman equation

$$Q^{\pi}\Big(x(t), u(t)|\theta^{Q}\Big) = c(x(t), u(t)) + \gamma Q^{\pi}\Big(x(t+1), u(t+1)|\theta^{Q}\Big). \tag{28}$$

To achieve stable performance in spite of the dynamics in the environment, we adopt the technique of the scatter network in the construction of a neural network. The scatter networks extract features from the input of the computation model and communication model, respectively. Then, the extracted special features are input into the core network to make decisions. Finally, the corresponding actions are output through different networks. Experimental results show that our scatter network structure can indeed achieve stable performance in a dynamic environment. First, in order to distinguish the different inputs between the computation state and the communication state, the policy network uses two multilayer perceptions (MLPs), which contains two fully connected layers, to extract the

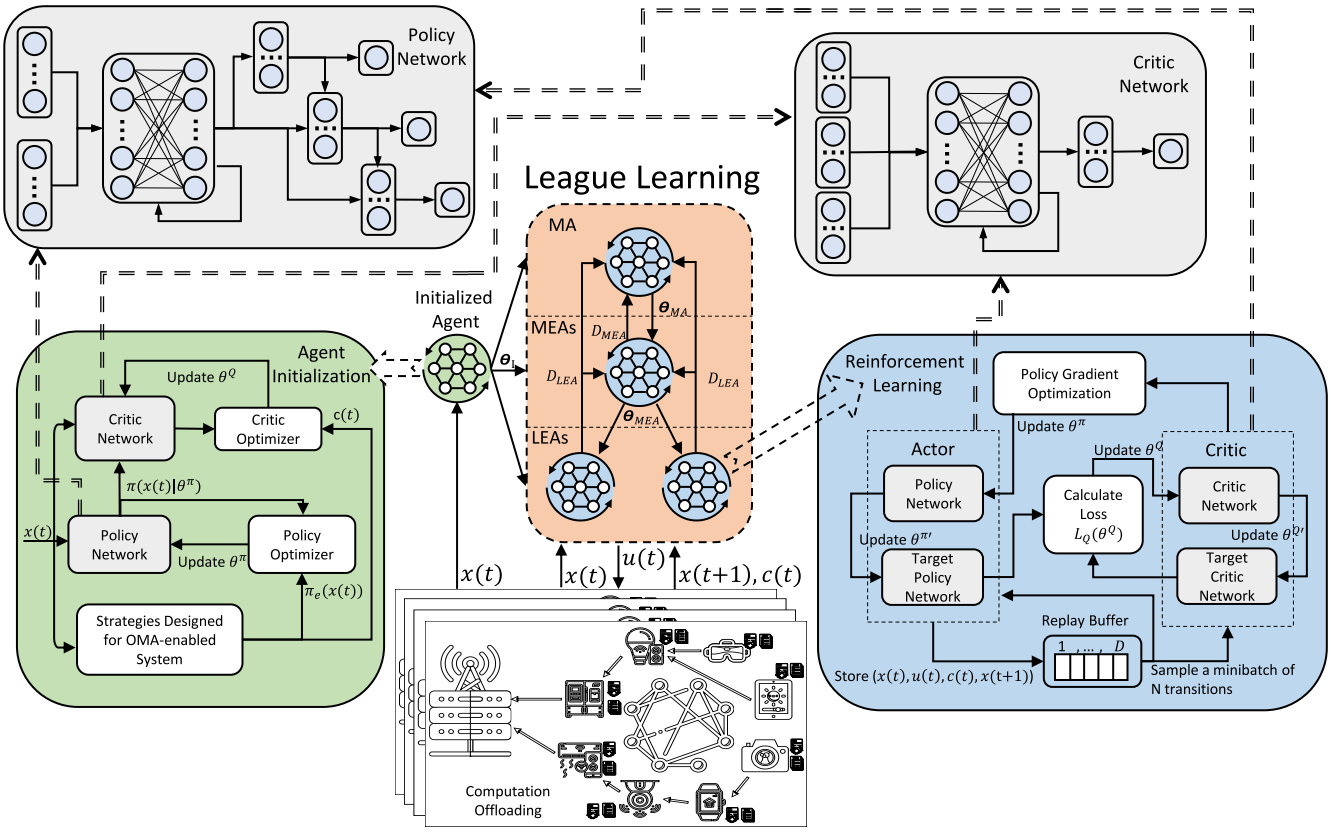

Fig. 2. League learning framework for cooperative computation offloading. The framework maintains a hierarchical league, where upper agents accept the experiences from lower agents for reinforcement learning, and the lower agents periodically update their network parameters using that of upper agents and initialization agent. The scatter connection structure of the agent's policy network and critic network are shown in the top of the figure.

features of the computation state and communication state, respectively. The critic network maintains three MLPs to handle inputs, where the extra MLP is added to extract the features of action. We then concatenate their outputs and let them be the input of the core of each network. The core of the network is a bidirectional long short-term memory (BiLSTM) [37], which maintains the memory between steps. With the use of BiLSTM, each device can perceive the features of the devices nearby. They are also affected by the features of the far away devices after attenuation. The output actions of the policy network are given by three MLPs. The agent first outputs the offloading decision. Then, the policy network outputs the distance decision depending on the offloading decision. Moreover, the agent decides the transmit power that should be allocated based on the previous offloading and distance decision. The output of the critic network is given by one MLP.

However, directly using the output of the policy network cannot make the agent to discover more strategies. We constructed an exploration policy $u'(t)$ by adding a noise interference as

$$u'(t) = \pi\left(x(t)|\theta^\pi\right) + \varepsilon_t \tag{29}$$

where the noise $\varepsilon_t \sim \mathcal{N}$ is sampled from a normal distribution and the variance of $\mathcal{N}$ annealing during the training phase.

B. Agent Initialization

Following the agent and network structure defined above, we next design the initialization method. The existing DRL-based algorithms applied in the MEC system usually cost a long training time to achieve convergence. We try to accelerate the convergence by avoiding useless exploration, which was caused by the agent's random initialization. Note that there are many state-of-the-art conventional methods that are applied to computation offloading problems, which provide lots of expert strategies for our agent to train. In order to avoid the early useless exploration caused by random initialization of network parameters, we initialize our agent using the strategies designed for the MEC system. Our agent starts with supervised learning on expert policy and output predictive action of IoT devices and the corresponding action value. We then compute the mean-square error between the expert's policy π_e and the agent's policy as

$$L_\pi\left(\theta^\pi\right) = \left(\pi\left(x(t)|\theta^\pi\right) - \pi_e(x(t))\right)^2 \tag{30}$$

where $x(t) \sim p_e(x(t))$ is the state sample from the state distribution under expert policy. During imitation learning process, the estimated gradient with respect to θ^π is calculated as

$$\nabla_{\theta^\pi} L_\pi = \left(\pi\left(x(t)|\theta^\pi\right) - \pi_e(x(t))\right)\nabla_{\theta^\pi} \log \pi\left(x(t)|\theta^\pi\right). \tag{31}$$

Then, the policy network θ^{π} is updated through the minibatch stochastic gradient descent method as follows:

$$\theta^{\pi} \leftarrow \theta^{\pi} + \alpha \nabla_{\theta^{\pi}} L_{\pi} \tag{32}$$

where α is the learning rate for the policy network. In order to avoid the agent's policy network overfitting expert policy, we impose L_2 regularization on the network parameters. Since the agent policy is constantly learning the expert policy, the critic network also needs to be constantly updated to suit the process of following RL. We define the loss function for updating the critic network as

$$L_Q\left(\theta^Q\right) = \Big(Q\Big(x(t), u(t)|\theta^Q\Big) - c(x(t), u(t)) + \gamma Q\Big(x(t+1), \pi\left(x(t+1)|\theta^p i\right)|\theta^Q\Big)\Big)^2. \tag{33}$$

Then, the estimated gradient with respect to θ^Q is calculated as

$$\nabla_{\theta^Q} L_Q = \Big(\gamma Q\Big(x(t+1), \pi\left(x(t+1)|\theta^{\pi}\right)|\theta^Q\Big) - c(x(t), u(t)) + Q\Big(x(t), u(t)|\theta^Q\Big)\Big)\nabla_{\theta^Q} Q\Big(x(t), u(t)|\theta^Q\Big). \tag{34}$$

Thus, we update the critic network θ^Q by minibatch stochastic gradient ascent as follows:

$$\theta^Q \leftarrow \theta^Q + \beta \nabla_{\theta^Q} L_Q \tag{35}$$

where β is the critic learning rate. We save the initialized parameters as θ_I^{π} and θ_I^Q for the following use of league. Algorithm 1 summarizes the implementation of agent initialization of the proposed algorithm.

Theorem 1: There exists a bound of distribution change while the initialized agent's policy π_I is close to the expert's policy π_e.

Proof: According to the agent's deterministic policy that $u(t) = \pi_I(x(t)|\theta^{\pi_I})$, π_e is close to π_I if $\pi_e(u(t) \neq \pi_I(x(t)|\theta^{\pi_I})|x(t)) \leq \epsilon$, where $\epsilon \in [0, 1]$. We have

$$p_e(x(t)) = (1-\epsilon)^t p_I(x(t)) + \left(1 - (1-\epsilon)^t\right) p_{\text{mistake}(x(t))} \tag{36}$$

where p_e and p_I is the state distribution of expert policy and initialized agent, p_{mistake} is the state distribution of mistake state. We have the distribution mismatch probability that

$$\begin{aligned} |p_e(x(t)) - p_I(x(t))| &= \left(1 - (1-\epsilon)^t\right)|p_{\text{mistake}}(x(t)) - p_I(x_t)| \\ &\leq 2\left(1 - (1-\epsilon)^t\right) \\ &\leq 2\epsilon t \end{aligned} \tag{37}$$

where we use the identity that $(1-\epsilon)^t \geq 1 - \epsilon t$ for $\epsilon \in [0, 1]$. So there exist a bound of distribution between p_I and p_e. ■

C. League Establishment

We then design a hierarchical league where the agents explore the environment collaboratively. There are three levels of training the agent in league learning, differing mainly in their mechanism for parameters reset. In the top level is the main agent (MA) that is used to evaluate the performance of the whole league. In the middle level and the third level, the main exploiter agent (MEA) and the league exploiter agent (LEA) are used to discover diverse and specialized strategies that capture the complexities of NCoCO. Comparatively, the MEA's performance is more stable than the LEA's. At the beginning of each trajectory, each exploiter agent (EA) runs its own policy for n steps in its environment. After n steps, each EA sends the trajectory through a queue to its upper level agents, which consists of states, actions, and rewards $s_1, a_1, r_1, \ldots, s_n, a_n, r_n$ together with the initial BiLSTM state. All agents use replay buffers that store trajectories for updating their network parameters asynchronously. The MA then continuously updates its network parameters on batches of trajectories received from MEAs and LEAs. MEAs update their policies on batches of trajectories collected by itself and receive from LEAs. Each LEA updates its policy on batches of trajectories collected by itself.

Algorithm 1: Agent Initialization

Initialization:
Initialize $\theta_I^{\pi}, \theta_I^Q, \pi_e(\cdot), \gamma, \alpha, \beta$
Iteration:
for epoch k in 1,2,... **do**
  Sample a random minibatch of N state $x(t)$ from distribution of expert strategy $p_e(x(t))$.
  Input $x(t)$ into policy network π_I and expert policy π_e.
  Calculate policy gradient as (31)
  Update the policy network θ_I^Q through stochastic gradient ascent as (32)
  Calculate critic estiamted gradient as (34)
  Update the policy network θ_I^Q through stochastic gradient ascent as (35)
Return: $\theta_I^{\pi}, \theta_I^Q$

The critic network parameters are updated by gradient descent on the loss function with respect to the target network, i.e., the critic loss is defined as

$$J_Q\left(\theta^Q\right) = \left(y_t - Q\Big(x(t), u(t)|\theta^Q\Big)\right)^2 \tag{38}$$

where y_t is calculated by target networks

$$y_t = c(x(t), u(t)) + \gamma Q'\Big(x(t+1), \pi'\Big(x(t+1)|\theta^{\pi'}\Big)|\theta^{Q'}\Big). \tag{39}$$

During the training phase, the estimated gradient with respect to θ^Q is calculated as

$$\nabla_{\theta^Q} J = (y_t - Q(x(t), u(t)))\nabla_{\theta^Q} Q\Big(x(t), u(t)|\theta^Q\Big). \tag{40}$$

The critic network is updated though a minibatch stochastic gradient descent method as follows:

$$\theta^Q \leftarrow \theta^Q - \beta \nabla_{\theta^Q} J_Q \tag{41}$$

where β is the critic learning rate.

We use policy gradient to update the policy network, which updates it in the direction of

$$\nabla_{\theta^{\pi}} J_{\pi} = \nabla_{\theta^{\pi}} \pi\left(x(t)|\theta^{\pi}\right) \nabla_u Q\Big(x(t), u|\theta^Q\Big)|_{u=\pi(x(t)|\theta^{\pi})}. \tag{42}$$

Then, the policy network is updated by the following stochastic estimated gradient method:

$$\theta^{\pi} \leftarrow \theta^{\pi} - \alpha \nabla_{\theta^{\pi}} J_{\pi} \tag{43}$$

where α is the learning rate of the policy network.

D. Populating the League

For the league established above, we populate the league by periodically resetting the networks of agents in the league to achieve robust performance. First, the MA resets its networks as the initialized agent at the beginning of the training phase as

$$\Theta_{\mathrm{MA}} \leftarrow \Theta_I \tag{44}$$

where $\Theta_I = [\theta_I^\pi, \theta_I^Q]$ are the network parameters of the policy network and critic network.

Second, MEAs regularly reset their networks with respect to both initialized agent's network and MA's network. Every T_{MEA}, MEAs reset their networks, i.e.,

$$\Theta_{\mathrm{MEA}} \leftarrow \tau_{\mathrm{MEA}}\Theta_{\mathrm{MA}} + (1-\tau_{\mathrm{MEA}})\Theta_I \tag{45}$$

where τ_{MEA} is the reset ratio of MEA.

Third, LEAs regularly reset their networks with respect to the initialized agent's network and MEAs' networks. Every T_{LEA}, LEAs reset their networks, i.e.,

$$\Theta_{\mathrm{LEA}} \leftarrow \tau_{\mathrm{LEA}}\Theta_{\mathrm{MEA}} + (1-\tau_{\mathrm{LEA}})\Theta_I \tag{46}$$

where τ_{LEA} denotes the reset ratio of LEA. We set $T_{MEA} = n$ and represent the number of LEAs within the league. To simplify the problem, we set $\tau_{\mathrm{MEA}} = \tau_{\mathrm{LEA}}$. However, much exploration is wasted when an EA's performance is always lower than that of MA. It is witty to reset such EA's network ahead of time. Moreover, the weight of the network parameters inheriting from the upper level agents will continuously increase during the training phase. As off-policy, each agent maintains a replay buffer to make efficient use of hardware optimizations, which means the agent learns in minibatches, instead of online. In our algorithm, the input dimension and the output dimension are determined by the number of IoT devices N. The number of hidden neurons in MLP and BiLSTM is set to H. The complexity analysis of MLP computing in the network is $O(NH)$ and the complexity analysis of BiLSTM is $O(H^2)$. Thereby, the complexity analysis is $O(NH + H^2)$ per decision slot. The pseudocodes of the league learning are shown in Algorithm 2.

Algorithm 2: League Learning

Initialization:
for agent $n \in \mathcal{L}$ **do**
  Initialize $\theta_n^\pi, \theta_n^{\pi'} \leftarrow \theta_I^\pi, \theta_n^Q, \theta_n^{Q'} \leftarrow \theta_I^\pi, \mathcal{D}_n, \tau \ll 1$.
Iteration:
for episode k in 1, 2, ... **do**
  for agent $n \in \mathcal{L}$ **do**
    Initialize a random process $\mathcal{N}$ for action exploration.
    Receive initial observation state $x(1)$
    for t= 1,...,T **do**
      Select exploration action $u'(t) = \pi(x(t)|\theta^\pi) + \mathcal{N}$ according to the current policy and exploration noise.
      Execute action $u'(t)$ and observe cost $c(r)$ and new state $x(t+1)$.
      Store transition $(x(t), u'(t), c(t), x(t+1))$ in D_n, and send them to its upper agent.
    for n and n's upper agents **do**
      Sample a random minibatch of N transition $(x(t), u(t), c(t), x(t+1))$ from D_n.
      Update critic network use the gradient calculated in (40)
      Update the policy network using the sampled gradient calculated in (42)
      Update the target network use soft update method:
      $\theta_n^{\pi'} \leftarrow \tau\theta_n^\pi + (1-\tau)\theta_n^{\pi'}$,
      $\theta_n^{Q'} \leftarrow \tau\theta_n^Q + (1-\tau)\theta_n^{Q'}$,
  if $k\%T_{LEA} == 0$ or $k\%T_{MEA} == 0$ **then**
    Reset agents as (45) or (46).

V. Simulation Result

A. Simulation Setup

We mainly consider the case of the number of IoT devices $N = 50$ and the number of edge servers $B = 1$.The coverage is set to 2×2 km^2. The maximum transmit power $p_{\max}$ is set to 1 W. There is also a switch power $E^{sw} = 0.01$ W for each relay's transmission. In our blockchain-empowered environment, the background noise σ^2 is set to -87 dBm. Considering the wireless interference model in the real world, we set the channel gain $g_{n,s} = d_{n,s}^{-\alpha}$, where $d_{i,s}$ is the distance between the device n and the offloading target device s, and $\alpha = 2.2$ is the path-loss factor.

In our computation model, the average computation capacities of the blockchain-empowered IoT devices and the edge servers are 1 and 30 GHz, respectively. The huge computing resource gap between local and edge makes IoT devices prefer to offload tasks to edge servers for execution, which is also more helpful for us to verify our multihop cooperative computation offloading learning algorithm. Time is divided into decision slots in our hybrid task model. At the beginning of each time slot, the average data size of the task is 3 GB. We set the average required CPU cycles per data size to 1 Gcycles per GB data. The maximum decision slot in each episode is set to $T = 32$. The system bandwidth is set to 20 MHz and the spectral efficiency is set to 10. For each IoT device n, the energy weight β_n^e is uniformly distributed in [0,1] and the delay weight β_n^d is set as $1 - \beta_n^e$.

The stepsizes are the parameters of alpha and beta in the proposed algorithm. The smaller the stepsizes, the slower the change of the network weight. In the experiments, we run the trials as many as possible to adjust the stepsizes and thus set the stepsizes alpha and beta to $1e-3$ that is a reasonable value ensuring our networks converging to an acceptable solution in an appropriate time. The smoothing constant is set to $\tau = 5e-3$. The discounting factor γ is set to 0.99. During the training phase, the agent initialization takes $4e4$ episodes and the league learning takes $1e6$ episodes. We maintain one MA, one MEA, and two LEAs within the league. The reset period of the league is set to 1/5 of the time that I-DDPG needs to achieve convergence and the reset ratio anneals from 100% to 0%. The proposed algorithm and other baseline methods are implemented using PyTorch on a Python-based simulator.

B. Simulation Design

Evaluation Metrics:

1) *Average Cost:* The average cost is calculated by the weighted ratio of the system cost of all local-computing

IoT devices to the total system cost. It can be used to evaluate the overall performance of all IoT devices.

2) *Number of Convergence Episodes:* The number of convergent episodes is that the learning algorithm could achieve convergence during the DRL-agent training process. In the NCoCO system, episodes can be converted into a specific clock-runtime.

Baseline Algorithms:

1) *Expert Algorithm [9]:* The expert algorithm proposed for the conventional OMA-enabled computation offloading system that formulates the joint problem as a potential game and achieves a state-of-the-art Quality-of-Service (QoS) performance.
2) *DDPG [38]:* DDPG, a single-agent DRL algorithm, is regarded as one of the state-of-the-art off-policy deep RL methods. The DDPG algorithm is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives.
3) *Initialization-DDPG (I-DDPG) Algorithm:* Intelligent agent represented by scatter networks first uses strategies designed for the OMA-enabled system for initialization and subsequently adopts DDPG to improve its performance through the policy gradient method.

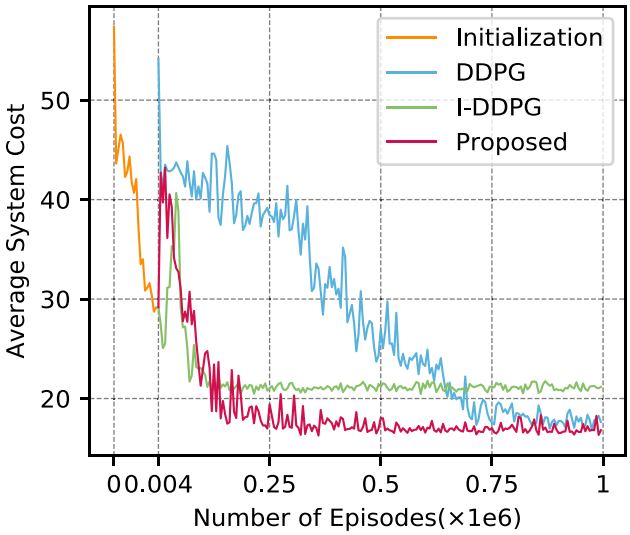

Fig. 3. Average cost versus the number of episodes.

C. Convergence Analysis

To demonstrate the convergence of our proposed learning algorithm and other baseline algorithms, we plot the dynamics of the average system cost over the episode in Fig. 3. We evaluate the convergence of the proposed learning algorithm using the above parameter settings. The convergence of the proposed learning algorithm is the crucial property for ensuring the obtained policy mapping states to the optimal actions. Generally. a learning algorithm is considered to converge when the average system cost curve becomes flat. Therefore, we plot the cost returned from each training episode under the scenario where the number of IoT devices is 50. As the result shown in Fig. 3, the initialized agent takes about $4e3$ episodes to achieve an approximate performance for the expert algorithm in the agent initialization phase. When it is in the reinforcement learning phase, it takes hundreds of episodes to achieve convergence. The DDPG algorithm spends $8.76e5$ episodes for convergence, and the I-DDPG algorithm takes $1.94e5$ episodes to achieve convergence. With the above league parameters setting, the proposed learning algorithm takes $3.18e5$ episodes to achieve convergence. In the beginning training phase of the DDPG algorithm, its average cost does not descend significantly. The DDPG algorithm prefers random exploration to find novel strategies. Its average cost is fluctuating, and then can be reduced gradually. Although the I-DDPG algorithm is the fastest one to reach a convergent state, its performance is not satisfactory. The reason is that there is not enough motivation for the intelligent agent after agent initialization to explore, so it quickly reaches a local-optimal performance and stops exploring. We can see that the curve of the proposed algorithm fluctuates more violently during the training of league, and then converges to the stable optimal solution of the system. Furthermore, the intelligent agent can achieve a convergence shortening at most 77.8% of the training time with agent initialization. In addition, the intelligent agent reducing 63.6% of the training time than the state-of-the-art reinforcement learning algorithm.

D. Performance Evaluation Under Various Environment Settings

We first compare the proposed learning algorithm with the baseline algorithms in Table II. We run our simulations with $N = 10, 20, \ldots, 50$ IoT devices and one edge server. For each algorithm, we repeated 200 times and computed the average system cost to give the result shown in Table II. Because of the large gap between the performance of the expert algorithm and other baseline algorithms shown in Table II, it is not shown in the subsequent experimental demonstration figures. When the number of industrial robots is 10, the performance of the algorithms is basically similar, and the proposed learning algorithm performs even a little prominently. However, with the increase in the number of IoT devices, our algorithm is gradually becoming outstanding. Furthermore, with the increase in the number of IoT devices, devices used in our NCoCO system have more choices in relay devices, so the unit cost of the system becomes even lower. This reflects the superiority of our NCoCO system. From Table II, we can see that the system optimal policy we finally learned can decrease the system cost by 45% compared with the expert policy. Even the I-DDPG algorithm can decrease 36.5% cost than the expert policy. Our proposed algorithm encourages the exploration of agent, hence the final convergence cost is better than that of the DDPG algorithm.

Figs. 4 and 5 show the average time cost and the average energy cost of different algorithms as the increasing numbers of IoT devices. The variation trends of the time cost or the energy cost are the same, with the increase in the number of devices. The performances of DDPG and the proposed algorithm are similar, and the latter is slightly better. The average

TABLE II
COMPARATIVE COST UNDER DIFFERENT NUMBERS OF IOT DEVICES

Cost / Algorithm / IoTs	Proposed	I-DDPG	DDPG	Expert
10 devices	4.72	5.08	4.78	5.77
20 devices	6.2	7.15	6.24	8.51
30 devices	7.64	9.27	7.9	12.46
40 devices	13.86	18.48	14.02	26.12
50 devices	16.62	21.17	17.2	30.23

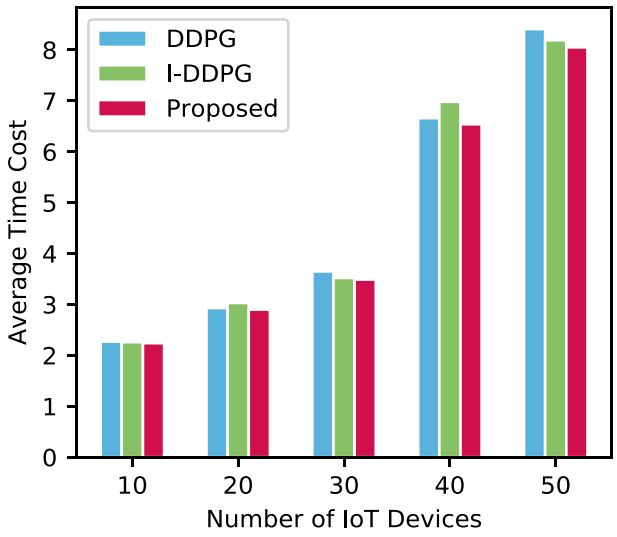

Fig. 4. Average time cost versus the number of IoTs.

time cost of the I-DDPG algorithm is similar to that of the other two algorithms, and even better than DDPG sometimes. But it has a huge gap with the other two in terms of average energy cost. The energy control of the I-DDPG algorithm is worse than that of DDPG and the proposed algorithm. The reason may be that the I-DDPG algorithm is affected by the expert policy designed for the OMA-enabled system. Note that the expert policy performs well in the OMA-enabled system.

Fig. 6 shows that the average cost of each algorithm increases with the increase in the data size of hybrid tasks. The larger data size will increase the average cost of the system under the same allocated bandwidth. Compared with the proposed algorithm and DDPG, the I-DDPG algorithm is more severely affected by the increase of data size. So that the I-DDPG algorithm is not good at controlling transmit power as we mentioned above. In addition, with the increase in the data size, the average of costs of the DDPG and I-DDPG algorithms gradually slow down.

Fig. 7 shows that the average cost decrease when the required CPU cycles per data size increase. Due to the limited computation resource of IoT devices, it is sensitive to the required CPU cycles per data size. When the required CPU cycles per data size are small, the performances of DDPG and our proposed learning algorithm are basically similar. When the required CPU cycles per data size increases, their performance will be distinguished. Thus, with the increase in the required CPU cycles per data size, IoT devices will be

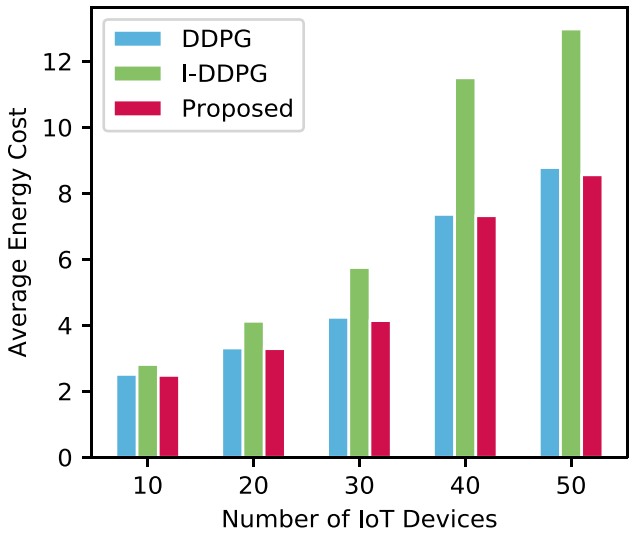

Fig. 5. Average energy cost versus the number of IoTs.

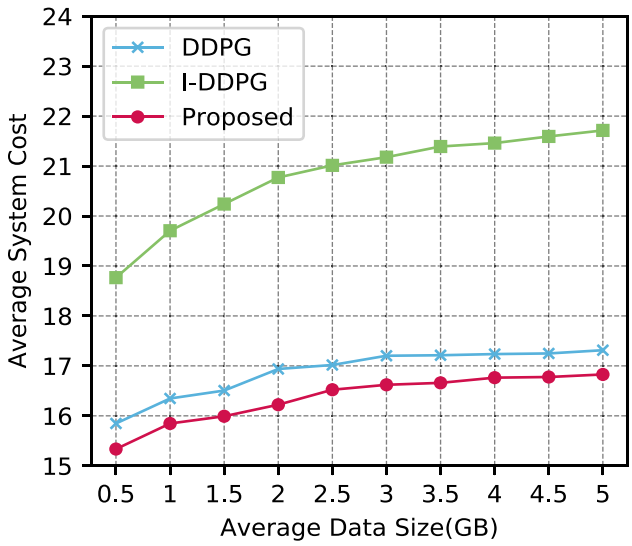

Fig. 6. Average system cost versus data size.

more inclined to offload tasks to the edge server for execution. As the computing resources of the edge server is also limited, the reduction in average system cost will gradually slow down.

We then present the performances under different computation capacities of IoT devices in Fig. 8. As the computation capacity of a single device increases, the relative benefits of offloading to the edge server become less. Because IoT devices prefer to execute their tasks locally. We also present the performances under different computation capacities of edge servers in Fig. 9. With an increase in the capacity of the edge server, the average cost of the system will not change significantly. Because more attention is paid to the energy control in the transmission phase and the selection of a relay device in our NCoCO system, the computing power of the edge side will not have too much impact on the average cost of the system. In particular, in different cases of computation capacities, the system cost of the proposed algorithm is less than that of DDPG and I-DDPG.

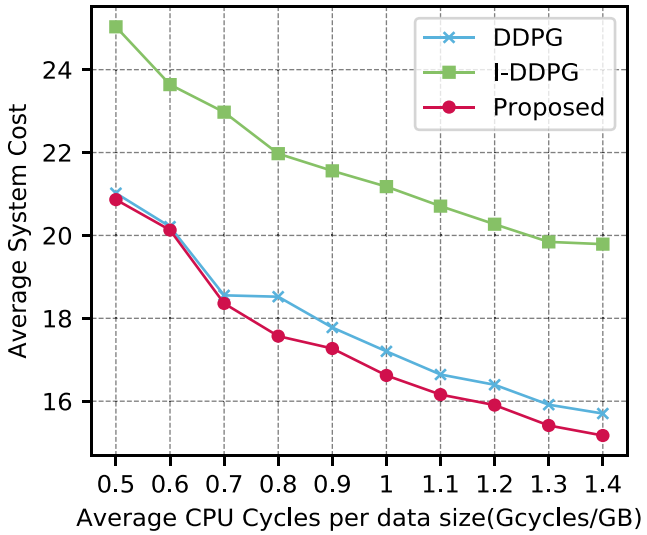

Fig. 7. Average system cost versus CPU cycles per data size.

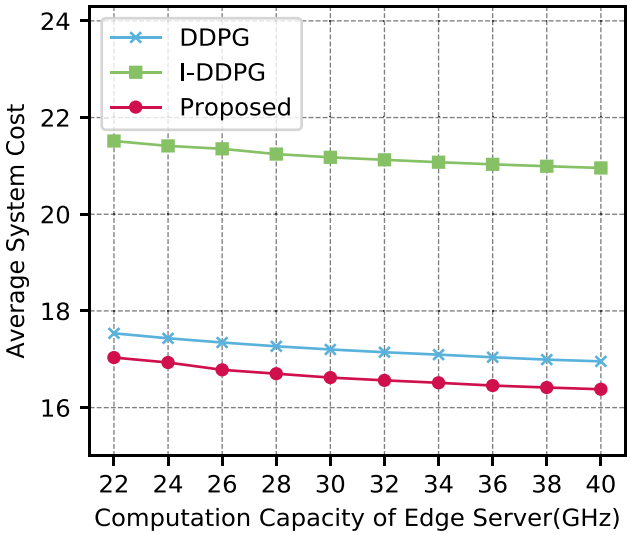

Fig. 9. Average system cost versus computation capacity of edge servers.

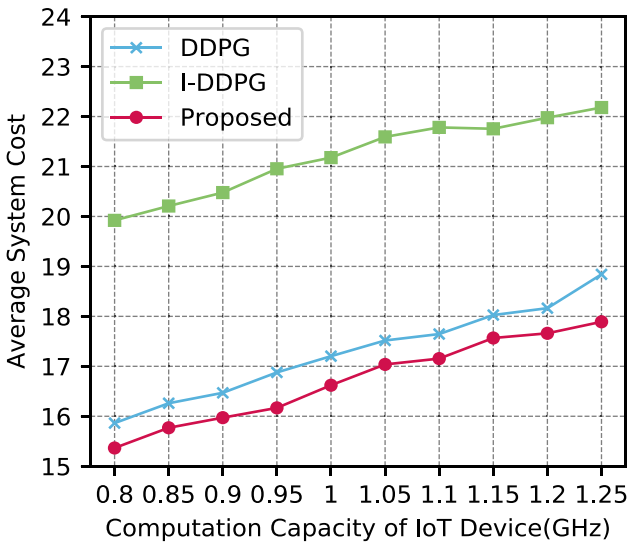

Fig. 8. Average system cost versus computation capacity of IoT devices.

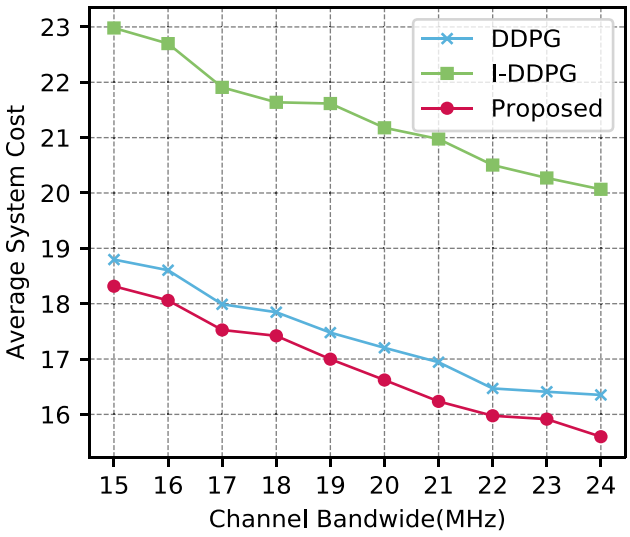

Fig. 10. Average system cost versus the channel bandwidth.

As illustrated in Fig. 10, the simulations run under scenarios where the channel bandwidth allocated in the system ranges from 15 to 24 MHz. As the channel bandwidth allocated by the system increases, the relative benefits of offloading to the edge server become less. In addition, because the selection of relay devices needs to be considered in the NCoCO system, the impact on the DDPG and the proposed learning algorithms is similar.

E. Impact of Different League Settings

In this section, we also consider the impact of different setting parameters in the league. The network of intelligent agents is initialized with the same network of agent initialization. Fig. 11 shows the performances under the different composition of the league, different reset period of agents, and different reset ratio of agents.

1) Different Compositions of League: We first investigate how the compositions of league affect the performance of the proposed algorithm and plot the results in Fig. 11(a) and (d). I-DDPG is equivalent to that there is only one MA in the league. Its number of episodes for convergence is minimum but the corresponding system cost is the worst. The proposed algorithm adds an MEA into the league to assist MA. Its number of episodes to achieve convergence increases, but the average cost of the system becomes smaller. Furthermore, the proposed algorithm adds one LEA into the league, which brings more diverse exploration to the league. Hence, the average cost of the system decreases significantly. However, in our simulation, adding two LEAs into the league cannot reduce the system average cost rapidly but increase the number of episodes.

2) Different Reset Periods of Exploration Agent: We then evaluate the achieved performances of the proposed algorithm under different reset periods, i.e., $T_{I-D}/2$, $T_{I-D}/5$, and $T_{I-D}/10$. We use the number of episodes T_{I-D} as a benchmark, which ensures the I-DDPG algorithm could achieve a convergence. As Fig. 11(b) shows, the average costs in

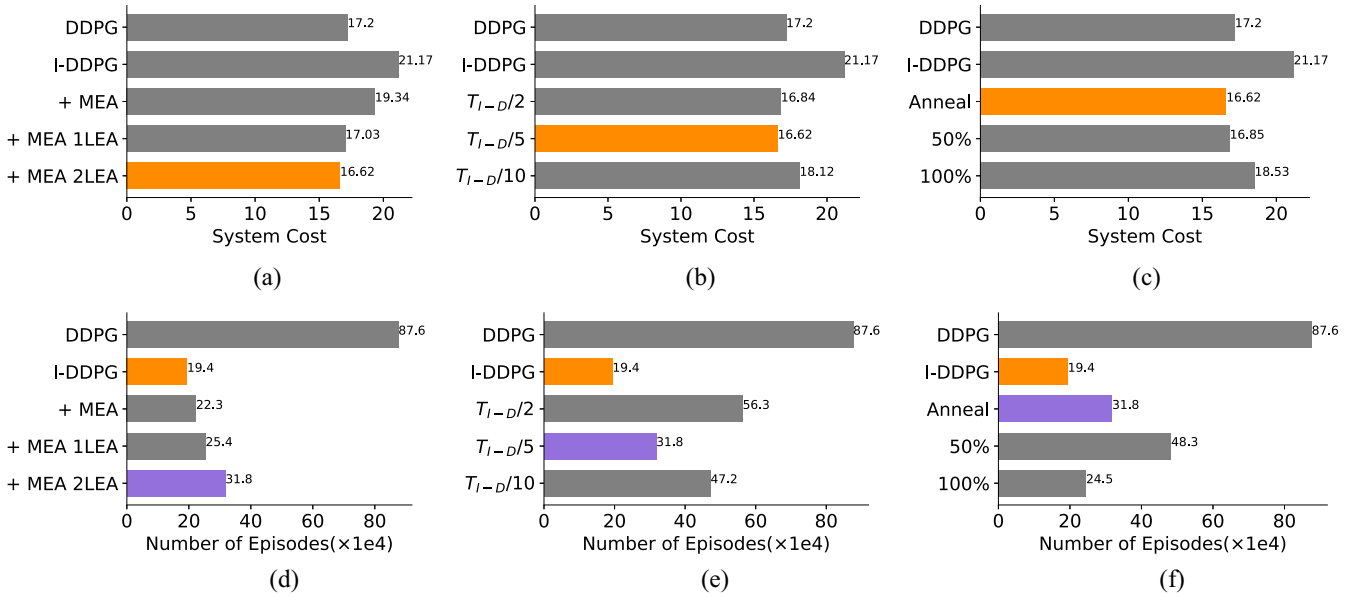

Fig. 11. Performance of League learning when varying of league parameters. (a) System cost versus league composition. (b) System cost versus reset period. (c) System cost versus reset ratio. (d) Number of episodes versus league composition. (e) Number of episodes versus reset period. (f) Number of episodes versus reset ratio.

$T_{I-D}/2$ and $T_{I-D}/5$ are similar and moreover the lowest. From Fig. 11(e), $T_{I-D}/2$ of convergence is nearly twice as much time as $T_{I-D}/5$. When the reset period is $T_{I-D}/2$, much time is wasted for exploration, whereas $T_{I-D}/5$ achieves a more effective exploration than $T_{I-D}/2$.

3) Different Reset Ratios of Exploration Agent: We record the results of the proposed algorithm under different reset ratios and plot them in Fig. 11(c) and (f). "Anneal" represents that the reset ratio anneals from 100% to 0%. At the beginning of learning, i.e., the reset ratio is 100%. EAs are used to explore the environment adequately. Then, the EAs inherit the experience of the upper agents in the later phase of the training. In this way, the league can prevent the previous experience of EAs exploration from being ignored by later populated agents. When the reset ratio is 0%, the league converges fastest but its performance is not satisfactory. Then, system costs in the cases of the reset ratio 50% and Anneal are similar. But the number of episodes in the case of the reset ratio 50% is a half and one of that in the case of Anneal.

VI. Conclusion

In this article, we proposed a novel multiagent DRL framework to minimize the long-term cost of cooperative computation offloading. To accelerate convergence, we trained our intelligent agent using expert strategies to balance exploration and exploitation. To enhance the robustness, we adopted league learning to allow hierarchical agents within a league to explore the environment collaboratively, so that the agent can discover more novel strategies and remain robust under different superparameters. To achieve stable performance in spite of the dynamics of the environment, we constructed scatter networks for the intelligent agent to distinguish information observed from the environment and actions. Simulation results demonstrated that the proposed algorithm has a more stable and robust performance than the state-of-the-art algorithms, and moreover, it could reduce almost 60% of training time whatever the system parameters change. In our future work, we will try to design the DRL-based framework for cooperative computation offloading with untrusted relays, to minimize information leakage at the untrusted relays and to achieve secure cooperative computation offloading. In addition, we plan to study the sparse representation in reinforcement learning to further improve the exploitation, which is a key problem for DRL.

References

[1] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.

[2] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. M. Leung, "Blockchain-based decentralized trust management in vehicular networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1495–1505, Apr. 2019.

[3] Z. Li, Z. Yang, S. Xie, W. Chen, and K. Liu, "Credit-based payments for fast computing resource trading in edge-assisted Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6606–6617, Aug. 2019.

[4] Z. Ma, L. Feng, and F. Xu, "Design and analysis of a distributed and demand-based backscatter mac protocol for Internet of Things networks," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 1246–1256, Feb. 2019.

[5] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.

[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," Sophia Antipolis, France, ETSI, White Paper, vol. 11, pp. 1–16, 2015.

[7] Z. Wang, Q. Zhao, L. Feng, and F. Xu, "How much benefit can dynamic frequency scaling bring to WiFi?" *IEEE Trans. Mobile Comput.*, early access, Dec. 9, 2019, doi: 10.1109/TMC.2019.2958323.

[8] H. Zhou, H. Wang, X. Li, and V. C. M. Leung, "A survey on mobile data offloading technologies," *IEEE Access*, vol. 6, pp. 5101–5111, 2018.

[9] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, "Qos-aware cooperative computation offloading for robot swarms in cloud robotics," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 4027–4041, Apr. 2019.

[10] Q.-V. Pham *et al.* (2019). *A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art*. [Online]. Available: https://arxiv.org/abs/1906.08452.
[11] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambrige, MA, USA: MIT Press, 1960.
[12] Z. Chen and X. Wang. (2018). *Decentralized Computation Offloading for Multi-User Mobile Edge Computing: A Deep Reinforcement Learning Approach*. [Online]. Available: https://arxiv.org/abs/1812.07394.
[13] C. Qiu, H. Yao, C. Jiang, S. Guo, and F. Xu, "Cloud computing assisted blockchain-enabled Internet of Things," *IEEE Trans. Cloud Comput.*, early access, Jul. 23, 2019, doi: 10.1109/TCC.2019.2930259.
[14] S. Zhou, H. Huang, W. Chen, Z. Zheng, and S. Guo. (2019). *Pirate: A Blockchain-Based Secure Framework of Distributed Machine Learning in 5g Networks*. [Online]. Available: https://arxiv.org/abs/1912.07860.
[15] Z. Xiong, S. Feng, W. Wang, D. Niyato, P. Wang, and Z. Han, "Cloud/fog computing resource management and pricing for blockchain networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4585–4600, Jun. 2019.
[16] Z. Xiong, Y. Zhang, N. C. Luong, D. Niyato, P. Wang, and N. Guizani, "The best of both worlds: A general architecture for data management in blockchain-enabled Internet-of-Things," *IEEE Netw.*, vol. 34, no. 1, pp. 166–173, Jan. 2020.
[17] Z. Xiong, J. Kang, D. Niyato, P. Wang, and H. V. Poor, "Cloud/edge computing service management in blockchain networks: Multi-leader multi-follower game-based admm for pricing," *IEEE Trans. Services Comput.*, vol. 13, no. 2, pp. 356–367, Mar./Apr. 2020.
[18] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16440–16455, 2020.
[19] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[20] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot-edge-cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.
[21] Z. Ning, X. Wang, J. J. Rodrigues, and F. Xia, "Joint computation offloading, power allocation, and channel assignment for 5g-enabled traffic management systems," *IEEE Trans. Ind. Informat.*, vol. 15, no. 5, pp. 3058–3067, May 2019.
[22] W. Chen *et al.*, "Cooperative and distributed computation offloading for blockchain-empowered industrial Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8433–8446, Oct. 2019.
[23] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.
[24] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, and S. Guo, "Green resource allocation based on deep reinforcement learning in content-centric IoT," *IEEE Trans. Emerg. Topics Comput.*, early access, Feb. 13, 2018, doi: 10.1109/TETC.2018.2805718.
[25] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
[26] J. Feng, F. R. Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile edge computing: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6214–6228, Jul. 2020.
[27] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994.
[28] A. Asheralieva and D. Niyato, "Learning-based mobile edge computing resource management to support public blockchain networks," *IEEE Trans. Mobile Comput.*, early access, Dec. 16, 2019, doi: 10.1109/TMC.2019.2959772.
[29] O. Vinyals *et al.*, "Grandmaster level in starcraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
[30] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.
[31] Y. Zhan, C. H. Liu, Y. Zhao, J. Zhang, and J. Tang, "Free market of multi-leader multi-follower mobile crowdsensing: An incentive mechanism design by deep reinforcement learning," *IEEE Trans. Mobile Comput.*, early access, Jul. 9, 2019, doi: 10.1109/TMC.2019.2927314.
[32] A. M. Haimovich, R. S. Blum, and L. J. Cimini, "MIMO radar with widely separated antennas," *IEEE Signal Process. Mag.*, vol. 25, no. 1, pp. 116–129, Dec. 2008.
[33] T. S. Rappaport *et al.*, *Wireless Communications: Principles and Practice*, vol. 2. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.
[34] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-aided computation offloading for trusted collaborative mobile edge computing," *IEEE Trans. Mobile Comput.*, early access, Aug. 14, 2019, doi: 10.1109/TMC.2019.2934103.
[35] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, 2012, pp. 2716–2720.
[36] B. Di, L. Song, and Y. Li, "Sub-channel assignment, power allocation, and user scheduling for non-orthogonal multiple access networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 11, pp. 7686–7698, Nov. 2016.
[37] Z. Huang, W. Xu, and K. Yu. (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging*. [Online]. Available: http://arxiv.org/abs/1508.01991.
[38] T. P. Lillicrap *et al.* (2015). *Continuous Control With Deep Reinforcement Learning*. [Online]. Available: http://arxiv.org/abs/1509.02971.

Zhenni Li (Member, IEEE) received the B.Sc. degree from the School of Physical Science and Electronics, Shanxi Datong University, Datong, China, in 2009, the M.Sc. degree from the School of Physics and Optoelectronic, Dalian University of Technology, Dalian, China, in 2012, and the Ph.D. degree from the School of Computer Science and Engineering, University of Aizu, Aizuwakamatsu, Japan, in 2015.

She is currently an Associate Professor with the Guangdong Key Laboratory of IoT Information Technology, School of Automation, Guangdong University of Technology, Guangzhou, China. Her research interests include machine learning, sparse representation, and resource allocation in cloud/edge computing.

Minrui Xu is currently pursuing the B.S. degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China.

His research interests include deep reinforcement learning, edge/cloud computing, Internet of Things, and blockchain.

Jiangtian Nie (Student Member, IEEE) received the B.Eng. degree (Hons.) in electronics and information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2016. She is currently pursuing the Ph.D. degree with ERI@N, Interdisciplinary Graduate School, Nanyang Technological University, Singapore.

Her research interests include incentive mechanism design in crowdsensing and game theory.

Jiawen Kang received the M.S. and Ph.D. degrees from the Guangdong University of Technology, Guangzhou, China, in 2015 and 2018, respectively.

He is currently a Postdoctoral Fellow with Nanyang Technological University, Singapore. His research interests mainly focus on blockchain, and security and privacy protection in wireless communications and networking.

Wuhui Chen (Member, IEEE) received the bachelor's degree from Northeast University, Shengyang, China, in 2008, and the master's and Ph.D. degrees from the University of Aizu, Aizuwakamatsu, Japan, in 2011 and 2014, respectively.

From 2014 to 2016, he was a Research Fellow with the Japan Society for the Promotion of Science, Tokyo, Japan. From 2016 to 2017, he was a Researcher with the University of Aizu. He is currently an Associate Professor with Sun Yat-sen University, Guangzhou, China. His current research interests include edge/cloud computing, cloud robotics, and blockchain.

Shengli Xie (Fellow, IEEE) received the M.S. degree in mathematics from Central China Normal University, Wuhan, China, in 1992, and the Ph.D. degree in control theory and applications from the South China University of Technology, Guangzhou, China, in 1997.

He is the Director of the Laboratory for Intelligent Information Processing and a Full Professor with the Guangdong University of Technology, Guangzhou. He has authored or coauthored two monographs and more than 100 scientific papers published in journals and conference proceedings. His current research interests include automatic control and signal processing, especially blind signal processing and image processing.