# Distributed Resource Scheduling for Large-Scale MEC Systems: A Multiagent Ensemble Deep Reinforcement Learning With Imitation Acceleration

Feibo Jiang, *Member, IEEE*, Li Dong, Kezhi Wang, *Senior Member, IEEE*, Kun Yang, *Senior Member, IEEE*, and Cunhua Pan, *Member, IEEE*

***Abstract*—In large-scale mobile edge computing (MEC) systems, the task latency, and energy consumption are important for massive resource-consuming and delay-sensitive Internet of Things Devices (IoTDs). Against this background, we propose a distributed intelligent resource scheduling (DIRS) framework to minimize the sum of task latency and energy consumption for all IoTDs, which can be formulated as a mixed-integer nonlinear programming. The DIRS framework includes centralized training relying on the global information and distributed decision making by each agent deployed in each MEC server. Specifically, we first introduce a novel multiagent ensemble-assisted distributed deep reinforcement learning (DRL) architecture, which can simplify the overall neural network structure of each agent by partitioning the state space and also improve the performance of a single agent by combining decisions of all the agents. Second, we apply action refinement to enhance the exploration ability of the proposed DIRS framework, where the near-optimal state-action pairs are obtained by a novel Levy flight search. Finally, an imitation acceleration scheme is presented to pretrain all the agents, which can significantly accelerate the learning process of the proposed framework through learning the professional experience from a small amount of demonstration data. The simulation results in three typical scenarios demonstrate that the proposed DIRS framework is efficient and outperforms the existing benchmark schemes.**

***Index Terms*—Distributed deep reinforcement learning (DRL), imitation learning, Lévy flight, multiagent reinforcement learning, resource scheduling.**

Manuscript received October 28, 2020; revised May 15, 2021, June 13, 2021, and August 13, 2021; accepted September 4, 2021. Date of publication September 20, 2021; date of current version April 25, 2022. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 41604117, Grant 41904127, Grant 61620106011, and Grant U1705263; in part by the Hunan Provincial Natural Science Foundation of China under Grant 2020JJ4428, Grant 2020JJ5105, and Grant 2021JJ30455; and in part by the Hunan Provincial Science Technology Project Foundation under Grant 2018TP1018 and Grant 2018RS3065. *(Corresponding authors: Li Dong; Kezhi Wang.)*

Feibo Jiang is with the Hunan Provincial Key Laboratory of Intelligent Computing and Language Information Processing, Hunan Normal University, Changsha 410081, China (e-mail: jiangfb@hunnu.edu.cn).

Li Dong is with the Key Laboratory of Hunan Province for New Retail Virtual Reality Technology, Hunan University of Technology and Business, Changsha 410205, China (e-mail: dlj2017@hunnu.edu.cn).

Kezhi Wang is with the Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K. (e-mail: kezhi.wang@northumbria.ac.uk).

Kun Yang is with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the School of Computer Sciences and Electrical Engineering, University of Essex, Colchester CO4 3SQ, U.K. (e-mail: kunyang@essex.ac.uk).

Cunhua Pan is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, London E1 4NS, U.K. (e-mail: c.pan@qmul.ac.uk).

Digital Object Identifier 10.1109/JIOT.2021.3113872

## I. Introduction

### A. Motivation

RECENTLY, with the rapid increase of resource-intensive tasks, e.g., augmented reality (AR), Internet of Things (IoT) applications, and autonomous driving, the quality of our life has the potential to be improved greatly. However, due to the limited size and battery life of IoT devices (IoTDs), these applications may be difficult to be implemented in practice. Fortunately, mobile edge computing (MEC) has been proposed recently as a promising technique to liberate IoTDs from computation-intensive tasks by allowing them to offload their high workloads to edge servers [1], [2].

However, due to the large number of IoTDs, one edge server may not be powerful enough to support all the devices at the same time. Thus, multiple MECs may be deployed to support the IoTDs. Then, it is critical to determine the offloading decision and resource allocation between computing and communication resource from different MECs to IoTDs [1]. Unfortunately, the above problem is normally formulated as a mixed-integer nonlinear programming (MINLP) problem [3], [4], which involves integer variables (i.e., offloading decision) and continuous variables (i.e., communication and computing resource allocation). This problem is very difficult to solve in general, especially in large-scale IoTD scenarios and dynamic environments.

### B. Background and Related Works

Some traditional solutions have been applied to solve the above problem, such as game theory [5], branch-and-bound method [6], and dynamic programming [7]. However, these solutions normally need a large amount of computing resource and it is difficult to realize online decision-making process.

Some other solutions, such as the convex relaxation [8] and heuristic local search [9] algorithms are also applied to handle the above problems. However, those algorithms normally need a considerable amount of iterations to achieve a satisfying local optimum, which may not be suitable for the dynamic environment. Moreover, with the increase of the number of IoTDs and MEC servers, the complexity of the above-mentioned traditional solutions increases significantly, which makes them very difficult to be applicable in a large-scale environment.

Fortunately, the emerging deep reinforcement learning (DRL) approach has shown great potential in solving the above-mentioned joint optimization problem [10], [11]. The DRL employs deep neural networks (DNNs) for mapping input MEC system parameters to the output resource scheduling decisions directly. After training from the environments, it can output the optimal decisions very quickly without explicitly solving the original mathematical optimization, which is very suitable for the fast varying and large-scale MEC environment. However, there are still several challenges yet to be addressed: 1) value-based DRL (e.g., $Q$-earning [12], deep $Q$ network (DQN) [13], [14] and double DQN [15]) can only work well in a limited action space, which is inefficient in large-scale application scenarios and 2) although policy-based DRL (e.g., deep deterministic policy gradient (DDPG) [16], advantage actor-critic (A3C) [17], and multiagent DDPG (MADDPG) [18]) can update policy by computing policy gradient for maximizing the expected reward, it is difficult to converge in complicated environment, especially for distributed architecture [18].

### C. Key Contributions

In this article, we aim to develop a distributed intelligent resource scheduling (DIRS) framework, which can be applied to large-scale MEC systems with multiple MECs and IoTDs in a dynamic environment. To enhance the performance of the framework, centralized training scheme is designed, whereas distributed decision making is proposed to increase the flexibility of the framework. The main contributions are summarized as follows.

1) We first formulate an optimization problem and the objective is to minimize the sum of task latency plus energy consumption for all the IoTDs. Then, we decompose the proposed MINLP optimization problem into an offloading decision subproblem and a communication and computing resource allocation subproblem, which can reduce the complexity of the original problem and guarantee that the solutions meet all the constraints.
2) We propose a distributed multiagent ensemble DRL framework for solving the decision-making subproblem. In this framework, one agent is deployed in each MEC to conduct the distributed decision making for IoTD offloading tasks to this MEC. To improve the performance of the whole system, we have a scheduler deployed in the Core-MEC (C-MEC) to conduct the centralized training with the global information in the training stage. Ensemble learning is introduced into this distributed DRL framework to simplify the neural network structure of each agent by partitioning the state space and improve the performance of a single agent by combining decisions of all the agents. Once the centralized training is completed, each agent in MEC can make distributed offloading decision only with local information.
3) We present a Lévy flight search as the action refinement to find the best actions for the DRL model according to the current state. Lévy flight search can help the DRL framework to skip the local optimum. In the Lévy flight search, $h$ mutation operator is used to generate mutant vector according to the channel state information, and then the Lévy crossover operator is applied to avoid candidate vector trapping into the local optimum. Finally, the greedy selection operator is applied to select the better solution between the candidate vector and original solution.
4) We propose an imitation acceleration scheme, which combines the DRL and imitation learning to accelerate the training process of all the agents. We first generate a small amount of demonstration data, and then we pretrain all the agents using a novel demonstration loss function, which can reduce the training time and increase the stability of the DRL-based framework.

### D. Organization

The remainder of this article is organized as follows. Section II presents a review of related works. Section III describes the system model and problem formulation. Section IV describes the detailed design of the DIRS framework. Section V presents the simulation results, followed by the conclusions in Section VI.

## II. Related Works

The DRL-based algorithms have attracted extensive attention in the resource scheduling field. In the following, we present the related works from four aspects: 1) value-based DRL; 2) policy-based DRL; 3) distributed DRL; and 4) Hybrid DRL.

*Value-Based DRL Method:* In [13], a deep $Q$-leaning-based offloading scheme was presented to optimize offloading policy according to the current battery level, the radio transmission rate, and the harvested energy in the MEC system. In [14], a DQN approach was introduced to optimize the networking, caching, and computing resources in the vehicular networks. Moreover, in [19], a deep reinforcement scheduling with experience relay is applied for mobile crowdsensing in fog computing.

*Policy-Based DRL:* In [16], a DDPG method was used to design the trajectory of unnamed aerial vehicles (UAVs) by jointly considering the communications coverage, fairness, energy consumption, and connectivity. In [17], a high-performance asynchronous A3C DRL algorithm is proposed to solve the complex dynamic resource allocation problem in vehicles network.

*Distributed DRL:* In [20], a novel two-timescale DRL (2Ts-DRL) was presented, which leveraged federated learning to train the 2Ts-DRL model in a distributed manner. In [21], a distributed DQN was applied to search the optimal subband and power level for transmission, and each vehicle-to-vehicle link was considered as an agent and trained in a decentralized way. Then, in [22], a decentralized offloading policy learned by DDPG for a multiuser MEC system was proposed, in which each DDPG was adopted to learn the efficient offloading policy interdependently at each mobile user. In [23], a meta-training enhanced multiagent DRL is introduced to design the trajectory of a team of drone base stations under unpredictable environments. In [24], the recent progress and future challenges of distributed learning in wireless networks are summarized.

*Hybrid DRL Method:* In [25], The task offloading method was proposed based on meta reinforcement learning, which can adapt fast to new environments with a small number of gradient updates and samples. In [26], an A3C DRL algorithm was introduced to obtain the resource pricing and allocation MEC enabled blockchain systems. In [27], the DRL with the Monte Carlo tree learning was introduced to solve the complex resource allocation problem for a collaborative MEC network. In [28], a generative adversarial network-powered deep distributional $Q$ network (GAN-DDQN) was proposed to learn the action-value distribution driven by minimizing the discrepancy between the estimated and the target action-value distribution, which can be used to solve the demand-aware resource allocation problem.

However, none of the above contributions considered the application of distributed DRL in online resource scheduling for large-scale MEC systems, which can be seen as a complex MINLP problem. Traditional value-based and policy-based DRLs are hard to be applied in large-scale MEC systems directly. For example, the value-based DRL can only work well in a limited action space, thus it may not be suitable for large-scale offloading decision-making environment. The advantage of policy-based DRL is that the outputs can be continuous, but it is easy to fall into the local optimal for the large action space and it is normally difficult to converge in dynamic environments [18]. Distributed DRL has a great potential in addressing the above-mentioned issues by conducting distributed learning and dynamic decision in large-scale application scenarios [22].

## III. System Model and Problem Formulation

### A. System Model

Fig. 1 shows our proposed system with one C-MEC and $M$ normal MECs, denoted as the set of $\mathcal{M} = \{1, 2, \ldots, M\}$. C-MEC is deployed at the macro base station, whereas the other edge servers are installed in each small base station. We assume there are $N$ IoTDs, denoted as the set of $\mathcal{N} = \{1, 2, \ldots, N\}$. Each IoTD has a computation task to be executed, which can be either offloaded to the MECs or processed locally.

We define the computing task in each IoTD as $U_i$, where $U_i = (F_i, D_i)\ \forall i \in \mathcal{N}$ [29], $F_i$ denotes the total number of the CPU cycles, and $D_i$ describes the data size transmitting to the MEC if offloading action is conducted. $D_i$ and $F_i$ can be obtained by using the approaches provided in [30] and [31].

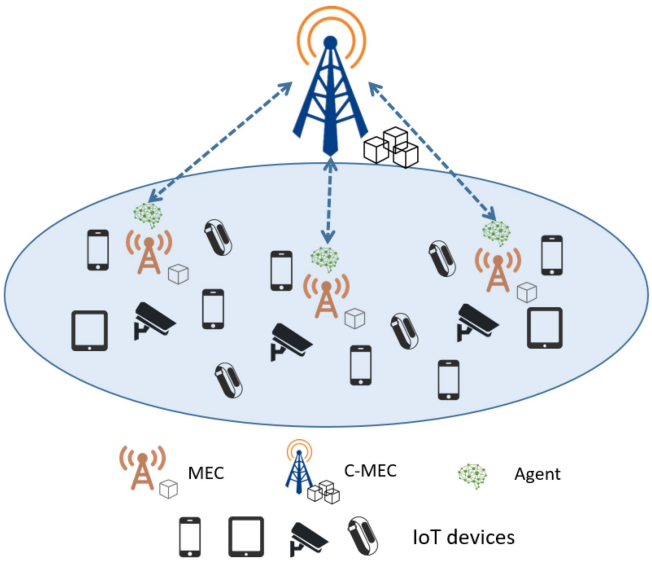



Fig. 1. Proposed MEC system.

Then, the overall time consumption of completing a task can be given by

$$T_{ij} = T_{ij}^{Tr} + T_{ij}^{C} = \frac{D_i}{r_{ij}} + \frac{F_i}{f_{ij}} \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M} \tag{1}$$

where $T_{ij}^{Tr}$ is time consumed for data offloading from the $i$th IoTD to the $j$th MEC, $T_{ij}^{C}$ is execution time in an MEC server if the $i$th IoTD offloading task to the $j$th MEC server, $f_{ij}$ is the computation capacity of the $j$th MEC allocating to the $i$th IoTD, and $j = 0$ if IoTD executes the task locally. Also, $r_{ij}$ denotes the offloading data rate from the $i$th IoTD to the $j$th MEC, which can be given by

$$r_{ij} = B \log_2\left(1 + \frac{p_{ij} h_{ij}}{\sigma^2}\right) \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M} \tag{2}$$

where $p_{ij}$ is the transmission power from the $i$th IoTD to the $j$th MEC server and $B$ is the channel bandwidth, $\sigma^2$ is the noise spectral density and $h_{ij}$ is the channel gain which is given by

$$h_{ij} = \frac{\beta_0 l_{ij}}{\left(X_j - x_i\right)^2 + \left(Y_j - y_i\right)^2} \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M} \tag{3}$$

where $\beta_0$ denotes the channel power gain at the reference distance, $l_{ij}$ describes the small-scale fading factor, $(x_i, y_i)$ is the coordinate of the $i$th IoTD, and $(X_j, Y_j)$ is the coordinate of the $j$th MEC.

We consider a binary offloading strategy as

$$a_{ij} = \{0, 1\} \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M} \tag{4}$$

where $a_{ij} = 1$ denotes that the $i$th IoTD decides to offload the task to the $j$th MEC, while $a_{ij} = 0$ denotes that the $i$th IoTD decides not to offload the task to the $j$th MEC. We assume that one IoTD can access to at most one edge server, which

is formulated as follows:

$$\sum_{j\in\mathcal{M}} a_{ij} + a_{i0} = 1 \quad \forall i \in \mathcal{N}. \tag{5}$$

Also, assume that the computing resource of each MEC is constrained by

$$\sum_{i=1}^{N} a_{ij} f_{ij} \le F_{j,\max}^{\mathrm{MEC}} \quad \forall j \in \mathcal{M} \tag{6}$$

where $F_{j,\max}^{\mathrm{MEC}}$ is the computational capability of the $j$th MEC.

Then, define $P_{i,\max}^{\mathrm{IoTD}}$ as the maximum transmission power that each IoTD can apply and then one has

$$\sum_{j=1}^{M} a_{ij} p_{ij} + a_{i0} p_i^E \le P_{i,\max}^{\mathrm{IoTD}} \tag{7}$$

where $p_i^E$ is the execution power of the $i$th IoTD which is given by $p_i^E = \kappa_i (f_{i0})^{v_i}\ \forall i \in \mathcal{N}$, and $f_{i0}$ is the local computing capacity and is a fixed value in this article, $\kappa_i \ge 0$ is the effective switched capacitance, and $v_i \ge 1$ is the positive constant. To match the realistic measurements, we set $\kappa_i = 10^{-27}$ and $v_i = 3$ [29].

### B. Problem Formulation

For each IoTD, the time consumption is

$$T_i = \sum_{j\in\mathcal{M}} a_{ij} T_{ij} + a_{i0} \frac{F_i}{f_{i0}}. \tag{8}$$

Also, for each IoTD, the energy consumption is given by

$$E_i = \sum_{j\in\mathcal{M}} a_{ij} \left( p_{ij} T_{ij}^{Tr} \right) + a_{i0} \frac{F_i}{f_{i0}} p_i^E. \tag{9}$$

Then, define $\Phi_i$ as

$$\Phi_i = \phi_T T_i + \phi_E E_i \tag{10}$$

where $\phi_T$ and $\phi_E$ are weighted coefficients. The weighted coefficients are determined by the specific application scenarios [32]. If the task is a time-sensitive task, one can set $\phi_T > \phi_E$. If the task is energy sensitive, then $\phi_T < \phi_E$ can be set. The weighted coefficients $\phi_T$ and $\phi_E$ regarding computational energy and time can be determined by applying the multiple criteria decision-making theory [33].

In this article, we aim to jointly optimize the offloading selection, computing resource allocation, and power allocation to minimize the weighted sum of task latency and energy consumption of all tasks. Specifically, we formulate the optimization problem as follows:

$$P0: \min_{\mathbf{a},\mathbf{f},\mathbf{p}} \sum_{i\in\mathcal{N}} \Phi_i \quad \text{s.t.} \quad (4)-(7) \tag{11}$$

where $\mathbf{a} = \{a_{ij} | i \in \mathcal{N}, j \in \mathcal{M}'\}$, and $\mathbf{f} = \{f_{ij} | i \in \mathcal{N}, j \in \mathcal{M}'\}$, $\mathbf{p} = \{p_{ij} | i \in \mathcal{N}, j \in \mathcal{M}'\}$ are vectors for offloading decisions, computing resource allocation, and transmission power from each IoTD, respectively. One can see that this problem includes both integer and continuous variables. If IoTD conducts the task itself, the energy consumption can be expressed as $p_{i0} = p_i^E$. Also assume that $\mathbf{h} = \{h_{ij} | i \in \mathcal{N}, j \in \mathcal{M}\}$ are time-varying input values, whereas other parameters are fixed.

### C. Problem Transformation

One can see that Problem $P0$ is an MINLP, which is very difficult to address in general. This problem becomes even more complex if it involves large-scale variables. To obtain the low complexity solution, we first decompose this problem into two subproblems: 1) offloading decision-making subproblem and 2) resource allocation subproblem.

First, we propose a distributed DRL framework to obtain optimal offloading decision $\mathbf{a}$ from the interaction between distributed agent and global environment. Once the offloading variable $\mathbf{a}$ is obtained, the resource allocation subproblem can then be expressed as follows:

$$P1: \min_{\mathbf{f},\mathbf{p}} \sum_{i\in\mathcal{N}} \left( \phi_T \left( \sum_{j\in\mathcal{M}} a_{ij} \left( \frac{D_i}{r_{ij}} + \frac{F_i}{f_{ij}} \right) + a_{i0} \frac{F_i}{f_{i0}} \right) + \phi_E \left( \sum_{j\in\mathcal{M}} a_{ij} p_{ij} \frac{D_i}{r_{ij}} + a_{i0} \frac{F_i}{f_{i0}} p_i^E \right) \right) \quad \text{s.t.} \quad (6), (7). \tag{12}$$

Then, we can decouple the above problem into the following optimization to facilitate the distribute decision making in each MEC as:

$$\min_{f_j, p_j} \sum_{i\in\mathcal{N}} \left( \phi_T \left( a_{ij} \left( \frac{D_i}{r_{ij}} + \frac{F_i}{f_{ij}} \right) + a_{i0} \frac{F_i}{f_{i0}} \right) + \phi_E \left( a_{ij} p_{ij} \frac{D_i}{r_{ij}} + a_{i0} \frac{F_i}{f_{i0}} p_i^E \right) \right) \quad \forall j \in \mathcal{M} \quad \text{s.t.} \quad (6), (7) \tag{13}$$

where $p_j = \{p_{1j}, \ldots, p_{Nj}\}$ and $f_j = \{f_{1j}, \ldots, f_{Nj}\}$.

The above problem only includes the continuous variables and therefore can be easily addressed using heuristic optimization methods (e.g., Lévy flight search).

## IV. Distributed Intelligent Resource Scheduling Framework

In this section, we will introduce the proposed DIRS framework, which focuses on joint computation offloading decision and resource allocation in the dynamic environment. Distributed model-free DRL is introduced to address the offloading decision-making problem, as it is a goal-oriented method which can learn the optimal policy through the interaction between agent and environment. In a large-scale MEC system with multiple users, there are several challenges to be addressed as follows: 1) the policy is randomly distributed and the experience replay buffer is sparse at the beginning of the learning process, so that the interaction process is inefficient and the DRL framework is difficult to converge, especially in dynamic situations; 2) because of the large number of users, the state space of the DRL is extremely large, which increases the difficulty of policy learning; and 3) the action exploration is very challenging because of

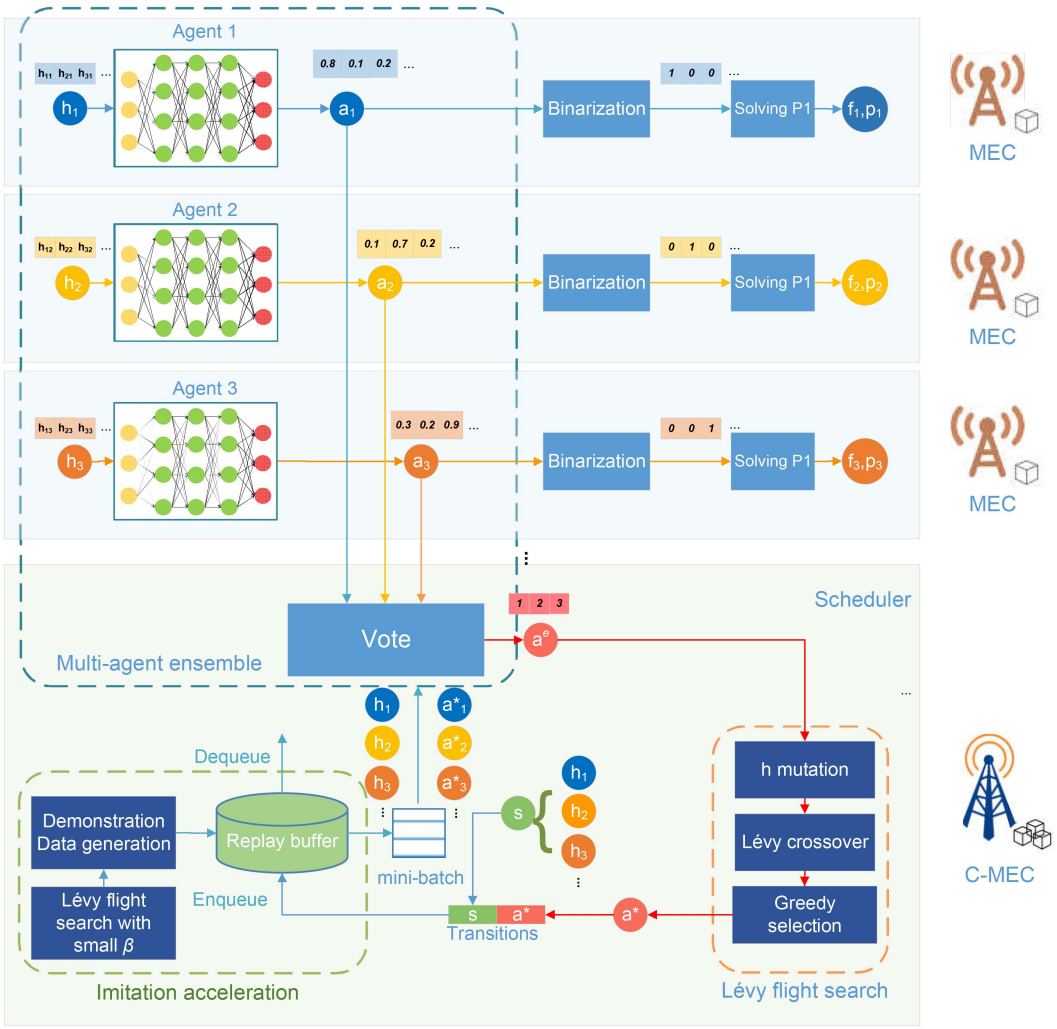



Fig. 2. DIRS framework.

the complex optimization problem such that the DRL is difficult to explore the optimal action and the search is prone to trap into local minimum. These challenges prohibit the DRL from being directly applied in the real environment. To tackle the large-scale decision problems, Google proposed an improved DRL with KNN as the action refinement in large discrete action spaces [34], and leveraged the demonstration data to accelerate the large-scale learning of DQN [35]. Inspired by the previous works, we present a DRL-based DIRS framework with centralized training and distributed inference. Next, we give a brief introduction of the proposed framework.

### A. Framework Outline

The DIRS framework is illustrated in Fig. 2, which includes local agents and a global scheduler. Specifically, the local agents are deployed on each MEC to conduct the independent decision making, and a global scheduler is deployed on the resource-abundant C-MEC to conduct the centralized training, which involves local information exchange of all the agents, generation of demonstration data by using the imitation acceleration scheme and refinement of the actions with the aid of Lévy flight search. In the online inference stage, each MEC only needs to perform some simple algebraic calculations with the help of a local agent enhanced by DRL instead of solving the original optimization problem.

The workflow of the DIRS framework is presented in Algorithm 1. First, we initialize parameters of all DNNs in $M$ agent with the parameters $\boldsymbol{\theta}^0 = \{\theta_1^0, \ldots, \theta_M^0\}$ randomly generated and we also initialize an empty replay buffer $\mathcal{D}$. Then, we pretrain all the agents using demonstration data produced by the imitation acceleration scheme (i.e., Algorithm 2) and obtain the pretrained parameters of all the agents $\boldsymbol{\theta}^D = \{\theta_1^D, \ldots, \theta_M^D\}$, and then keep all the demonstration data in the replay buffer $\mathcal{D}$. Next, the online training stage and online distributed inference stage are executed. Particularly, the online distributed inference stage is performed continuously, where each agent $j$ generates an offloading action $a_{j,t} = \{a_{1j,t}, \ldots, a_{Nj,t}\}$ for $N$ IoTDs according to the channel state information $h_{j,t} = \{h_{1j,t}, \ldots, h_{Nj,t}\}$ and policy $\pi$ with parameters $\theta_{j,t}$ at time slot $t$, and obtains $p_{j,t} = \{p_{1j,t}, \ldots, p_{Nj,t}\}$ and $f_{j,t} = \{f_{1j,t}, \ldots, f_{Nj,t}\}$ by solving Problem (13) independently. The online training stage is performed at every interval $\phi$ by the multiagent ensemble

**Algorithm 1** DIRS Framework

**Input:** $h_{j,t}, \phi$.

**Output:** $a_{j,t}, p_{j,t}, f_{j,t}$.

Initialize $M$ agents with policies randomly parameterized by $\boldsymbol{\theta^0}$.
Initialize an empty replay buffer $\mathcal{D}$.
**Offline pre − training stage**
Train all agents using demonstration data by **Algorithm 2**, and obtain the demonstrated parameters of all agents $\boldsymbol{\theta^D}$.
Keep all demonstration data in the replay buffer $\mathcal{D}$.
**Online distributed inference stage**
Set iteration number of online inference $T_{DRL}$.
**while** $t < T_{DRL}$ **do**
  **for** each agent $j$ **do**
    Obtain the channel state information $h_{j,t}$ according to the environment.
    Generate the offloading action $a_{j,t} = \pi\left(h_{j,t}|\theta_{j,t}\right)$ independently.
    Obtain $p_{j,t}$ and $f_{j,t}$ by solving Problem (13) independently.
  **end for**
  **Online training stage**
  **if** $t \bmod \phi = 0$ **then**
    Train all agents using **Algorithm 3** and **Algorithm 4**.
  **end if**
**end while**

algorithm (i.e., Algorithm 3) and Lévy flight search algorithm (i.e., Algorithm 4), and then the learned parameters of all $M$ agents $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_M\}$ are updated. $\phi$ is the training interval. These two stages are alternately performed and the offloading policies of all the agents can be gradually improved in the iteration process.

In general, the DIRS includes three work stages: 1) offline pretraining stage is applied to accelerate the DRL training for the large-scale application scenarios; 2) online training stage is introduced to track the variations of the real scenarios in dynamic environments; and 3) online distributed inference stage is presented to make real-time decisions. Moreover, there are three key improvements of the DIRS framework compared to traditional DRL.

1) An imitation acceleration scheme is presented to generate demonstration data and initialize the parameters of distributed agents rather than initializing them randomly to accelerate the learning speed of the distributed DRL (i.e., Section IV-B).
2) The multiagent ensemble algorithm is proposed in Section IV-C for large state space partition and decision consolidation, in which the original channel state information $\mathbf{h}$ is regarded as the current state $s$ of DRL and it is divided into smaller subsets $\{h_1, h_2, \ldots, h_M\}$ according to the ownership of MECs. Finally, the maximum vote approach is applied to integrate the results of all distributed agents, and obtain the ensemble offloading decision $\mathbf{a^e}$ according to the maximum vote. This method can realize dimensionality reduction for the DNN in each agent and simplify the policy learning in each substate space.
3) A novel Lévy flight search is introduced in Section IV-D for action refinement, which can enhance the action exploration of DRL. Then the optimal offloading action $\mathbf{a^*}$ is achieved by maximizing the reward which is cached into the replay buffer $\mathcal{D}$. One can see that the DIRS framework is a model free DRL which can provide distributed decision making and resource allocation without solving the original MINLP problem. We describe the implementation details of each module in the following sections.

### *B. Imitation Acceleration Scheme*

In large-scale scenarios, distributed DRL typically requires to learn a huge amount of data before they reach reasonable performance, which is very time consuming by trial and error. This is the major drawback of the DRL to solve the large-scale optimization problem. Recently, imitation learning has been shown to help address this difficult exploration problems in DRL [35], [36].

Imitation learning focuses on imitating human learning or expert demonstration for controlling the behavior of the agent, which can help DRL reduce the time required to learn by an agent to a great extent through reducing the number of trials [37]. In DRL, imitation learning can help an agent to achieve better performance in a complex environment by pretraining it with the demonstration data.

In the proposed DRL, we propose an imitation acceleration scheme combined with DRL and imitation learning. The imitation acceleration algorithm leverages relatively a small amount of demonstration data to pretrain the agents in our framework, which can significantly accelerate the learning process of the distributed DRL. The details of the imitation acceleration scheme are described as follows.

First, we collect demonstration data by leveraging the optimization algorithm for solving the problem in (11). In general, the algorithm can be divided into three categories: 1) if the action space is small, we can use the exhaustive search approach to obtain the optimal decision; 2) if the action space is medium, we can use some mixed-integer programming solver (e.g., CPLEX); and 3) if the action space is large, we can use some global heuristic algorithms to obtain suboptimal decisions [36]. In our study, the demonstration data is generated from the Lévy flight search with a small value of parameter $\beta$ (to be introduced in Appendix) which is suitable for solving large-scale MINLP problems and can possibly achieve globally optimal solutions [38]. Then, the channel state information as well as its optimal offloading actions solved by the Lévy flight search are stored into the replay buffer $\mathcal{D}$.

Second, we redefine the demonstration loss function from [35]. Because our DRL is not a standard DQN network, so we ignore the DQN loss terms in the original loss function. Then, the simplified demonstration loss function has two losses

$$L_1(\theta) = L_D(\theta) + \lambda_1 L_{L2}(\theta) \tag{14}$$

**Algorithm 2** Imitation Acceleration Algorithm

**Input:** $K$.

**Output:** pre-trained parameters $\boldsymbol{\theta}^D$.

1: Generate demonstration data and record them in the replay buffer $\mathcal{D}$.
2: Set iteration number of imitation learning $T_D$.
3: **while** $t < T_D$ **do**
4: Sample a minibatch of $K$ transitions from replay buffer $\mathcal{D}$.
5: Train all agent and update the offloading policy $\pi_t$ with parameters $\boldsymbol{\theta}^D$ using the demonstration loss in Eq. (14).
6: **end while**

where $L_D(\theta) = -(1/K)\sum_{k\in\mathcal{K}}((a_k^*)^T \log(\pi(h_k|\theta)) + (1-a_k^*)^T \cdot \log(1-\pi(h_k|\theta))$ is the demonstration data loss which is the cross-entropy for demonstration data and $K$ is the number of demonstration data in the batch, whereas $L_{L2}(\theta) = \|\theta\|^2$ is the $L2$-norm of $\theta$, which can increase the generalization of DNN in each agent.

Third, we initialize the pretraining process of all the agents solely on the demonstration data before starting any interaction with the environment.

Finally, once the offline pretraining phase is complete, all the agents start to act in the environment. Then, the online training stage and distributed inference stage are alternately executed. We describe the whole process of Algorithm 2 as follows.

### C. Multiagent Ensemble-Based Distributed DRL Algorithm

In the proposed large-scale MEC systems with a large number of IoTDs, the traditional DRL suffers from the challenge that the state space is also extremely large, and therefore it is difficult to train a steady policy because of the partial observability of the large state space. To address this, we introduce ensemble learning to enhance the distributed DRL as follows.

Ensemble learning is a machine learning method that generates and combines multiple inducers to solve the complex optimization problem. The intuitive explanation behind ensemble learning stems from human nature and the tendency to gather different opinions and combine them to make a complicated decision [39]. There are many advantages to introduce ensemble learning into our DIRS framework as follows.

1) *Global Optimization:* A single agent that conducts local inference may get stuck in a local optimum. By combining several agents, ensemble methods can decrease the risk of obtaining a locally minimal solution.
2) *Dimensionality Reduction:* Each agent can be constructed and trained using a selected subset of all the features (i.e., its own channel gains), which can reduce the impact of the curse of dimensionality by reducing the state space for each agent.
3) *Output Manipulation:* The original offloading decision can be seen as a multiclass classification problem for multiple MECs, but each agent is a binary classifier. By applying ensemble learning, many binary classifiers can be combined into a multiclass classifier, which can simplify the decision process of each agent.

To this end, we propose a multiagent ensemble algorithm to support the distributed DRL, in which the basic components are redefined as follows.

1) *State Space:* The overall channel state information is $\mathbf{h} = \{h_{ij}\}\ \forall i \in \mathcal{N}\ \forall j \in \mathcal{M}$. We also assume that each agent in each MEC can only obtain its own related channel state information, and therefore the state of agent $j$ is denoted as $h_j = \{h_{1j}, \ldots, h_{Nj}\}$.
2) *Action Space:* We define two kinds of action, i.e., global action and local action for each agent. Specifically, global action is applied in the training stage, whereas local action is generated in each agent for the distributed decision making. The global action is defined as $\mathbf{a^e} = \{a_i\}\ \forall i \in \mathcal{N}$, where $a_i = 0$ means the $i$th IoTD decides to execute the task itself, whereas $a_i = j$ means that the $i$th IoTD decides to offload the task to the $j$th MEC, where $j \in \mathcal{M}$. The local decision in the $j$th agent is defined as $a_j = \{a_{1j}, \ldots, a_{Nj}\}$, where $a_{ij} = 1$ means that the $i$th IoTD decides to offload the task to the $j$th MEC, while $a_{ij} = 0$ means the $i$th IoTD decides not to offload the task to the $j$th MEC. One can see that each agent can make distributed decision according to its own local information.
3) *Reward:* The global reward is defined as the reciprocal of the objective function in (11).
4) *Policy:* The policy $\pi_j(h_j|\theta_j)$ of each agent $j$ is implemented by applying a DNN, where $\theta_j$ is the parameters of the DNN at the $j$th agent.
5) *Vote:* The output of each agent can be combined by voting solutions. The final solution is chosen based on the highest value of the decisions of all the agents. Here, we give an example to illustrate the proposed voting process. Suppose there are three MECs and the outputs of three agents for the $i$th IoTD are $a_{i1} = 0.7$, $a_{i2} = 0.8$, and $a_{i3} = 0.2$, respectively. Since the second value $a_{i2}$ is the highest number, we can have the global ensemble action of the $i$th IoTD $a_i^e$ as 2, which means this IoTD will offload the task to the second MEC. If all the values of the $i$th IoTD are lower than 0.5, then we set the corresponding ensemble action $a_i^e$ of the $i$th IoTD as 0.
6) *Prioritized Experience Replay:* The scheduler in C-MEC refines the global action by applying the Lévy flight search (to be introduced in Section V-D) and adds the self-generated data to its replay buffer $\mathcal{D}$. Data will be added to the replay buffer until it is full, and then the agent starts to over-write old data in this buffer during the online training stage. Prioritization batch is also used in the replay buffer. The probability of sampling transitions $i$ is defined as

$$P_i = \frac{w_i^\tau}{\sum_{k\in\mathcal{K}} w_k^\tau} \tag{15}$$

where $w_i = |\Delta\delta_t| + \epsilon$ and $\epsilon$ is small positive constant that guarantees that all the transitions can be sampled. Two different $\epsilon_A$ and $\epsilon_D$ can be applied to control the relative sampling of the agent generated data versus the

**Algorithm 3** Multiagent Ensemble Algorithm

**Input:** $h_t, a_t, \mathcal{D}$.
**Output:** $\theta_t$.

```
Integrate the global a_t^e by voting form all agents.
Find the best a_t^* by Algorithm 4.
Append the state-action pair {h_t, a_t^*} to the replay buffer D.
Sample a random minibatch of transitions by priority strategy using Eq. (15) from replay buffer D.
Feed these transitions to all agents.
Set iteration number of distributed learning T_D.
for each agent j do
    while t < T_M do
        Update agent parameters θ_{j,t} by minimizing the loss function in Eq. (16) in distributed way.
    end while
end for
```

demonstration data. $\Delta\delta_t$ is the average loss function variation of all the agents. The exponent $\tau$ determines the intensity of prioritization.

7) *Loss Function:* The loss function of the DNN in the agent is proposed as follows:

$$L_2(\theta) = L_1(\theta) + \lambda_2 L_A(\theta) \tag{16}$$

where $L_1(\theta)$ is the demonstration loss function defined in (14) and $L_A(\theta)$ is the agent generated data loss which is given by

$$L_A(\theta) = -\frac{1}{S}\sum_{k\in S}\Big(\left(a_k^*\right)^T \log(\pi(h_k|\theta)) + \left(1-a_k^*\right)^T \log(1-\pi(h_k|\theta))\Big) \tag{17}$$

where $S$ is the number of agent generated transitions in the batch.

When we initialize the environment and obtain the state, the offline training stage is performed, in which each agent $j$ generates the offloading action $a_{j,t}$ according to the policy $\pi_{j,t}(h_{j,t}|\theta_{j,t})$ at the time slot $t$. Then, we obtain the global offloading decision $a_t^e$ according to the maximum vote of all the agents. To increase efficiency of action exploration, we search the best $a_t^*$ by applying Algorithm 4, and then we append the state-action pairs $\{h_t, a_t^*\}$ to the replay buffer $\mathcal{D}$ as training samples of all the agents. Next, we sample a batch of transitions by applying the priority strategy discussed before, and then we train all the agents using the Adam algorithm [40] and update all the offloading policies $\pi_t = \{\pi_{1,t}, \ldots, \pi_{M,t}\}$ by minimizing the loss function defined in (16). We describe the whole process of Algorithm 3 as follows.

### D. Lévy Flight Search

Action exploration is an important part in DRL. Traditional DRL normally uses the random process for action exploration (e.g., $\epsilon$-greedy), which is blind and inefficient in large action space [18]. Some local search methods are also applied for action exploration which can achieve the suboptimal offloading policy $\pi$ [34], [41]. However, these local search methods are easily stuck in local optimum and the right offloading policy cannot be guaranteed, especially in large-scale MEC systems [41]. To address this problem, we introduce a novel Lévy flight search, which can jump out of local optimum with high efficiency, to find the best offloading action $a_t^*$ according to $a_t^e$ in DRL. After Lévy flight search, the newly generated state-action pairs $\{s_t, a_t^*\}$ at time slot $t$ are appended to the replay buffer as training transitions of all the agents. Traditional Lévy flight is used to generate a new solution in the heuristic search as follows:

$$x_i(G) = x_i(G-1) + \eta d_i \tag{18}$$

where $x_i(G)$ is the solution at iteration $G$, $d_i$ is a Lévy flight step, and $\eta$ is a scale weight.

However, there are two disadvantages that avoid Lévy flight from being directly applied to our distributed DRL algorithm. First, the search step $d_i$ output from Lévy flight is a continuous real number, but the offloading decision $a_i$ in our problem is an integer value. Second, it does not take advantage of the channel state information in the search process. To tackle these issues, we propose a novel Lévy flight search, in which the solution can be represented as

$$x = \left[a_1, \ldots, a_N, f_1, \ldots, f_N, p_1, \ldots, p_N\right] \tag{19}$$

where $N$ is the number of IoTDs, $a_i$, $f_i$, and $p_i$ denote the offloading decision, allocated computation resource, and transmission power of the $i$th IoTD, respectively.

The Lévy flight search consists of three operations: 1) $h$ mutation; 2) Lévy selection; and 3) greedy selection. We introduce each of them as follows.

1) *$h$ Mutation:* channel state information **h** provides the prior information to create a mutant vector $a_i^m(G)$. The $h$ mutation is applied by comparing the normalized $h_{ij}$ with a Lévy flight step, which represents that the IoTD that offloads the task to the MEC with higher $h_{ij}$ has a higher stability. Thus, the $h$ mutation for the integer part of the solution can be represented as follows:

$$a_i^m(G) = \begin{cases} \text{randm}, & \text{if } \gamma d_i > \frac{h_{i,a_i}}{\sum_{j\in\mathcal{M}} h_{ij}} \\ a_i(G-1), & \text{otherwise} \end{cases} \tag{20}$$

where randm $\in \mathcal{M}'$ is a randomly generated integer to ensure that the $i$th IoTD will offload the task to an MEC or execute the task itself, $\gamma$ is a decreased weight calculated as follows:

$$\gamma = 2 - 2G/G_{\max} \tag{21}$$

where $G_{\max}$ is the maximum iteration number.
The rest part of the solution [e.g., $f_i^m(G)$ and $p_i^m(G)$] is generated by (18).

2) *Lévy Crossover:* The Lévy crossover is carried out to produce a candidate vector $x_i^c(G)$ by combining the mutant vector $x_i^m(G)$ and a target vector $x_i(G-1)$, which is achieved by comparing the weighted Lévy flight step with a threshold $th$. Thus, the Lévy crossover is represented as follows:

$$x_i^c(G) = \begin{cases} x_i^m(G), & \text{if } \gamma d_i > th \\ x_i(G-1), & \text{otherwise.} \end{cases} \tag{22}$$

**Algorithm 4** Lévy Flight Search Algorithm

**Input:** $a_t^e, \beta, th$

**Output:** $a_t^*$

1: Initialize $x(0)$ with ensemble offloading decision $a_t^e$, and random computation resource and energy resource allocation.
2: **while** $G \leq G_{max}$ **do**
3: Generate Lévy search step $d_i$.
4: Generate a mutant vector $x_i^m(G)$ with $a_i^m(G)$ by Eq. (20)-Eq. (21), and $f_i^m(G)$ and $p_i^m(G)$ by Eq. (18) in h mutation.
5: Obtain a candidate vector $x_i^c(G)$ by Lévy crossover in Eq. (22).
6: Calculate the fitness of $x_i^c(G)$ and $x_i(G-1)$ by solving the objective function in Eq. (11).
7: Select the subsequent solution $x_i(G)$ by greedy selection in Eq. (23).
8: Update $\gamma$ according to Eq. (21).
9: **end while**

3) *Greedy Selection:* The selection operator determines whether the candidate vector $x_i^c(G)$ or the target vector $x_i(G-1)$ survives into the next iteration. The greedy selection is used to select the vector with the better fitness as follows:

$$x_i(G) = \begin{cases} x_i^c(G), & \text{if } f\left(x_i^c\right) < f(x_i(G-1)) \\ x_i(G-1), & \text{otherwise} \end{cases} \quad (23)$$

where $f(\cdot)$ denotes the objective function in (11).

In a word, the Lévy flight search employs the channel state information to guide the mutation of a solution, and introduces Lévy steps to avoid solution trapping into the local optimum, so it is an efficient action exploration method. The detailed description of the Lévy flight search algorithm is provided in Algorithm 4.

In addition, the variation factor $\beta$ (to be introduced in Appendix) is the key parameter of a Lévy flight to balance global and local search. If $\beta$ value is large, the step size of a Lévy flight will be restricted in a small search range, which can really focus on the local search and occasionally global search. Therefore, larger $\beta$ value implies the faster convergence speed and slightly lower solution accuracy. If the $\beta$ value is small, the search length of walking distance of a Lévy flight is long so that the global search can be enhanced and the optimal solution is achieved with slow convergence speed. For these above reasons, we use a Lévy flight search with a large $\beta$ value in online training stage (i.e., Ensemble learning at Algorithm 3) and apply a Lévy flight search with small $\beta$ value in offline pretraining stage (i.e., Imitation acceleration at Algorithm 2).

### *E. Time Complexity Analysis of DIRS Framework*

There are two key elements which will influence the time complexity of our DIRS framework. They are the DNN learning and the Lévy flight search. The time complexity of DNN learning is $O(K \times I_{\text{DNN}} \times T_N)$, where $K$ denotes the number of samples, $T_N$ is the iteration number of DNN learning, and $I_{\text{DNN}}$ indicates the time complexity of one iteration, which depends on the number of parameters in DNNs. The time complexity of a Lévy flight search is $O(P \times I_{LF} \times G_{\max})$, where $P$ denotes the population size, $G_{\max}$ is the maximum iteration number, and $I_{LF}$ indicates the time complexity of the update of one feasible solution, which depends on the dimension of the feasible solution and the evaluation complexity of the feasible solution [42].

So, in our DIRS, the time complexity of the offline pretraining stage (Imitation learning) is $O(M \times K \times I_{\text{DNN}} \times T_D)$, where $M$ indicates the number of MECs and $T_D$ is the maximum iteration number for imitation learning. The offline pretraining stage is carried out in the core MEC or the remote cloud with abundant resources. In multiagent ensemble learning, the time complexity of the online training stage for the core MEC is $\max(O(K \times I_{\text{DNN}} \times T_M), O(P \times I_{LF} \times G_{\max}))$, while the time complexity of the online training stage for the normal MEC is $O(K \times I_{\text{DNN}} \times T_M)$. Considering the MEC system with limited resources, the $K$ and $T_M$ of the DNN and the $P$ and $G_{\max}$ of the Lévy flight in the online training stage are set with very small values. The time complexity of the online distributed inference stage is $O(I_{\text{DNN}})$ for each MEC, which is a low time complexity and is suitable for IoT and MEC system.

TABLE I
SIMULATION PARAMETERS

| Parameters | Assumptions |
|---|---|
| Data size of task $D_i$ | 100kB |
| Required CPU cycles of task $F_i$ | $10^9$ cycles |
| Bandwidth $B$ | 1MHz |
| Local Computational Capability $f_{i0}$ | $10^9$ cycles/s |
| Maximum transmission power $P_{max}^{IoTD}$ | 1.5W |
| Noise Spectral Density $\sigma^2$ | $10^{-12}$W/Hz |

## V. SIMULATION RESULTS AND NUMERICAL ANALYSIS

### *A. Simulation Environment Settings*

We first present experimental settings of the MEC system in Table I. The proposed DIRS framework is implemented on the Python 3 platform. The parameters of the DIRS framework are chosen as follows: replay buffer size = 1024, minibatch size = 256, and training interval $\phi = 10$. The parameters of the imitation acceleration scheme are chosen as follows: $T_D = 500$, $\lambda_1 = 10^{-4}$. The parameters of the multiagent ensemble algorithm are chosen as follows: $T_M = 2, \lambda_2 = 0.5, \epsilon_A = 0.08$, and $\epsilon_D = 0.02$. The parameters of the Lévy flight search are chosen as follows: $P = 5$, $G_{\max} = 10$, $\beta = 1.5$, and $th = 0.4$. All simulations are carried out in Tensorflow 2.2 environment running on a i7-6500U CPU with 16-GB RAM and 512-GB SSD. The agents of the DRL are designed by Keras using the dense layer and Adam optimizer. Next, we present two different evaluations to verify the performance of the DIRS framework.

TABLE II
PERFORMANCE COMPARISON OF IMITATION LEARNING

| Data quantity | $L_1(\theta)$ | | $L_D(\theta)$ | |
|---|---|---|---|---|
| | Training accuracy | Testing accuracy | Training accuracy | Testing accuracy |
| 64 | 90.8% | 90.1% | 90.3% | 85.5% |
| 128 | 92.4% | 91.8% | 92.2% | 89.8% |
| 256 | 96.7% | 96.3% | 96.6% | 93.4% |
| 512 | 96.7% | 96.5% | 96.6% | 94.3% |

### B. Performance Evaluation for Different Modules of DIRS

In this section, we consider the performance for different modules of DIRS in a medium-scale IoT scenario. There are two MEC servers and 30 IoTDs randomly distributed in the squared area with size 50 m $\times$ 50 m. We assume that the coordinates of two MEC servers are (10 m, 10 m) and (40 m, 40 m). The remote computational capability of MEC $F_{\max}^{\text{MEC}}$ is set to 15 $\times 10^9$ cycles/s. The 4-layer fully connected DNNs are applied as the agents, which includes 30, 80, 60, and 30 neurons in each layer, respectively. We collect 3000 environment data as the state inputs of DRL and $T_{DRL}$ is set to 3000.

Imitation learning is used as a pretraining tool in our DIRS framework, and the amount of demonstration data will influence the performance of pretraining. We compare the performance of imitation learning with various quantities of demonstration data using different loss functions in Table II. It can be stated that, the DNNs may not improve its learning performance significantly when demonstration data quantity is above 256. Moreover, the proposed loss function $L_1(\theta)$ can achieve better training and testing accuracy than $L_D(\theta)$. The reason behind this is when demonstration data quantity is above 256, the over-fitting issue may happen. Therefore, the $L_2$ regularized term is added in $L_1(\theta)$ and it can ensure the generalization of DNN, which can lead to a higher learning accuracy, especially for the testing process.

Multiagent ensemble learning assisted DRL is the core part of the DIRS framework, which uses multiagent learning and decision consolidation. Fig. 3 shows the convergence curves of all the agents in different MECs. One can see that the loss values of Agent 1 and Agent 2 all converge to 0.2 after around 300 iterations, which means all the agents in our DIRS framework can work properly and the over fitting does not happen in ensemble learning.

Moreover, a Lévy flight search is applied to refine the action in the proposed framework. Fig. 4 characterizes the action refinement performance using a classic Lévy flight search and our proposed Lévy flight search. It can be observed from Fig. 4 that the proposed Lévy flight search achieves better performance with lower fitness value than the classic Lévy flight search. The reason of higher accuracy of the proposed Lévy flight search is that the channel state information is applied to guide the action exploration and the Lévy flight crossover is applied to jump out from the local optimum during the search process.

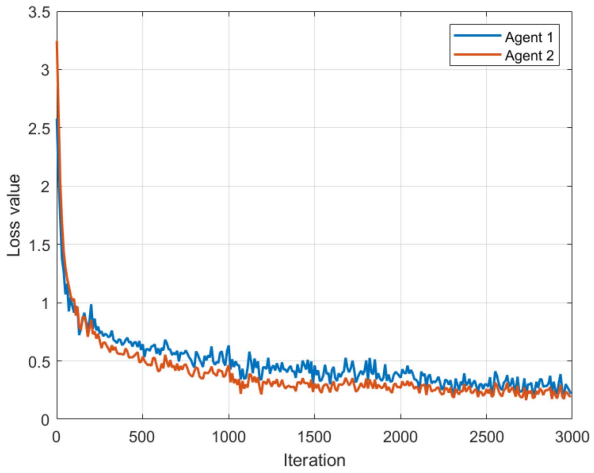



Fig. 3. Comparison of loss values for Agent 1 and Agent 2.

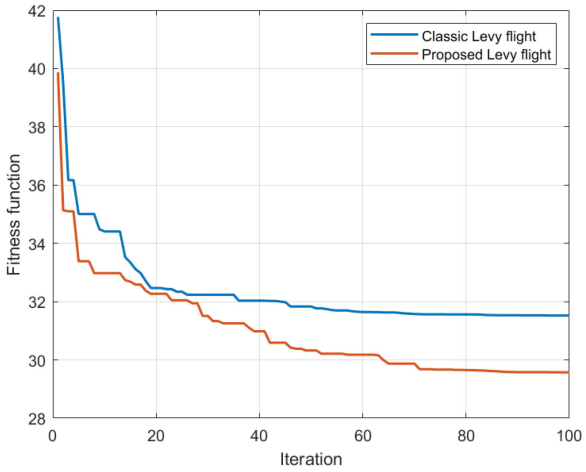



Fig. 4. Action refinement using different Lévy flight searches.

Next, we analyze the benefits of a Lévy flight search and imitation acceleration for the whole DIRS framework by using the performance metric of the average reward during the training process. One can see from Fig. 5 that the ensemble DRL with a Lévy flight search converges to a higher average reward than the ensemble DRL with the $\epsilon$-greedy search. This is due to the fact that the Lévy flight search is a heuristic local search which can refine the action and jump out of the local optimum by Lévy steps. One can also see that the DIRS with imitation acceleration converges faster than the ensemble DRL without imitation acceleration in Fig. 5.

### C. Performance Evaluation for the Large-Scale IoT Scenario With Different Parameters

In this section, we consider the performance of DIRS in a large-scale IoT scenario. Assume that there are eight edge servers and 180 IoTDs, randomly distributed in the squared area with size 50 m $\times$ 50 m. The coordinates of eight edge servers are obtained by the $k$-means algorithm for all the IoTDs at the first time. The remote computational capability of MEC $F_{\max}^{\text{MEC}}$ is set to 10 $\times 10^9$ cycles/s. The 4-layer fully connected DNNs are applied for the agents, which includes 180, 260,

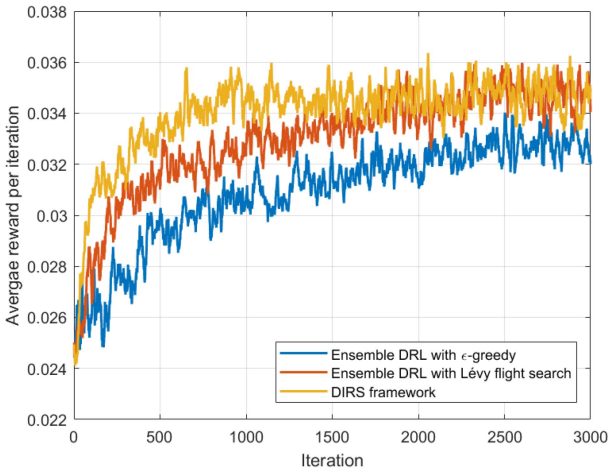



Fig. 5. Comparison of average reward curve.

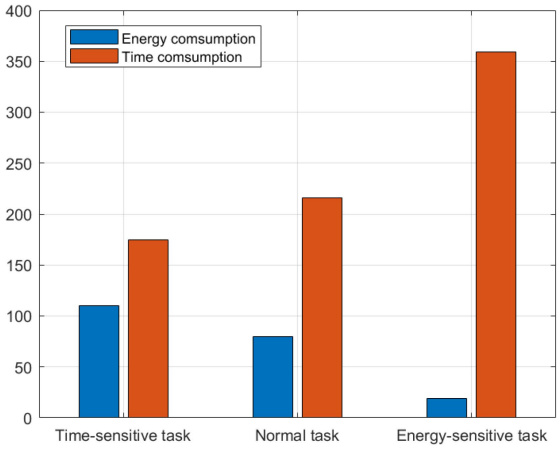



Fig. 6. Comparison of the DIRS framework with different weighted coefficients.

200, and 180 neurons in each layer, respectively. We collect 4000 environment data as the state inputs of DRL and $T_{DRL}$ is set to 4000.

We first evaluate the performance of the DIRS framework with different weighted coefficients in (10). We define three different tasks: 1) time-sensitive task ($\phi_T = 1.5$ and $\phi_E = 0.5$); 2) normal task ($\phi_T = 1$ and $\phi_E = 1$); and 3) energy-sensitive task ($\phi_T = 0.5$ and $\phi_E = 1.5$). Fig. 6 characterizes the energy and time consumption of all the tasks. It can be observed that the time consumption decreases gradually when the $\phi_T$ increases. Additionally, the energy consumption decreases with the increase of $\phi_E$, as expected.

Then, we evaluate the performance of the DIRS framework with different position distributions of IoTDs. We select three different random functions to generate the positions of IoTDs, which are uniform distribution, Gaussian distribution, and Lévy distribution. Table III characterizes the objective function values of the DIRS framework under different distributions of IoTDs. For evaluating the influence of the different position distributions of IoTDs, we define the normalized reward rate (NRR), which is equal to that the inferred reward dividing the optimal reward [43]. In NRR, the inferred reward in the numerator is calculated from the output of the DIRS framework, and the optimal reward in the denominator is obtained from the particle swarm optimization (PSO) algorithm which is always applied to solve large-scale MINLP problems with high quality but low efficiency [43].

TABLE III
PERFORMANCE OF THE DIRS FRAMEWORK WITH DIFFERENT POSITION DISTRIBUTIONS OF IOTDS

| Metric | Uniform distribution | Gaussian distribution | Lévy distribution |
|---|---|---|---|
| Objective function | 295.27 | 291.07 | 290.38 |
| Time consumption | 215.84 | 203.76 | 201.83 |
| Energy consumption | 79.43 | 87.31 | 88.55 |
| NRR | 0.9536 | 0.9528 | 0.9574 |

TABLE IV
PERFORMANCE OF THE DIRS FRAMEWORK WITH DIFFERENT $D_i$ AND $F_i$

| Metric | $D_i$ | | $F_i$ | |
|---|---|---|---|---|
| | 50kB | 1000kB | $0.5\times10^9$ cycles | $2\times10^9$ cycles |
| Objective function | 290.54 | 304.89 | 147.11 | 587.99 |
| Time consumption | 200.37 | 225.52 | 107.77 | 395.72 |
| Energy consumption | 90.17 | 79.37 | 39.34 | 192.27 |
| NRR | 0.9517 | 0.9519 | 0.9524 | 0.9508 |

From Table III, one can see that the IoTDs with nonuniform distributions (e.g., the Gaussian distribution and Lévy distribution) obtain lower objective function values. One can also find that all the distributions achieve similar NRR values, which means the DIRS framework can work efficiently under different position distributions.

Next, we evaluate the influence of different data size $D_i$ and required CPU cycles $F_i$ in the DIRS framework. We select two different $D_i$ when $F_i = 10^9$ cycles, and two different $F_i$ when $D_i = 100$ kB. Table IV characterizes the objective function values and NRRs of the DIRS framework with different $D_i$ and $F_i$. It can be observed that when $D_i$ or $F_i$ increase, the time to complete the task and the energy consumption also increase, as expected. Moreover, the DIRS framework achieves good performance for different $D_i$ and $F_i$, which means the DIRS framework is insensitive to the task parameters if it has been trained adequately.

### *D. Performance Evaluation for the Large-Scale IoT Scenario With Different Benchmarks*

In this section, we consider the performance of DIRS in a large-scale IoT scenario. There are ten MEC servers and 200 IoTDs randomly distributed in the squared area with size 50 m × 50 m. The coordinates of ten MEC servers are

TABLE V
OFFLOADING PERFORMANCE COMPARISON OF DIFFERENT OFFLOADING SCHEDULING STRATEGIES

| Metric | Training time | Inference time | Average time consumption | Average energy consumption |
|---|---|---|---|---|
| DIRS | 434.23 | 0.0163 | 292.8779 | 40.5198 |
| AC | 489.64 | 0.017 | 322.7262 | 53.4802 |
| DDPG | 517.17 | 0.0198 | 302.8238 | 47.2354 |
| PSO | - | 1.8565 | 293.3282 | 40.8271 |
| DE | - | 1.8699 | 293.7273 | 40.6938 |

obtained by the $k$-means algorithm for all IoTDs at the first time. The remote computational capability of MEC $F_{\max}^{\mathrm{MEC}}$ is set to 10 $\times 10^9$ cycles/s. The 4-layer fully connected DNNs are applied as the agents, which includes 200, 300, 250, and 200 neurons in each layer, respectively. We collect 5000 environment data as the state inputs of DRL and $T_{DRL}$ is set to 5000.

We first compare the DIRS framework with two well-known DRL algorithms and two heuristic algorithms.

1) Actor-critic (AC) is a combination of the actor-only DRL and the critic-only DRL. The critic uses an critic's DNN and simulation to learn a value function, which is then used to update the policy of the actor's DNN.
2) DDPG is a state-of-the-art AC, which includes two critic's DNNs and two actor's DNNs for performance enhancement.
3) PSO is a heuristic search which is suitable for solving large-scale MINLP problems for offloading optimization and can normally achieve near-optimal global solutions but with long computation time [44].
4) Differential evolution (DE) is another latest heuristic search for task scheduling optimization in large-scale MEC systems [45].

Table V characterizes the training time, inference time, and average time and energy consumption of all offloading methods for online joint resource scheduling. It can be observed that the DIRS framework achieves the lowest average time consumption and average energy consumption while consuming the least training time. The superiority of the DIRS framework can be attributed to three aspects: 1) imitation learning from the demonstration data accelerates the training process of DIRS; 2) ensemble learning simplifies the structure of DNN by state space partition, which leads to less learning time of each agent, while decision consolidation according to the global information improves the reward of DIRS; and 3) Lévy flight search refines the action and enhances the exploration, which leads to efficient search and fast convergence. It can also be seen the DIRS framework has the shortest inference time, which can be explained for the following reasons: 1) the DIRS is a distributed DRL framework, in which the trained agents can make offloading decisions in a parallel way and 2) each agent just needs to solve the local optimization problem in (13), which is simpler than the original optimization problem. Moreover, the DIRS framework provides almost the same average energy and time consumption as PSO and DE, while PSO and DE take 113.89 and 114.72 times longer inference time than the DIRS framework, respectively.

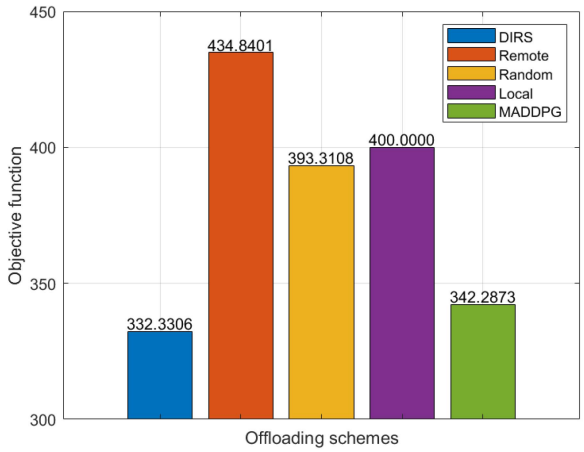



Fig. 7. Comparison of the object function with different offloading schemes.

The inference performance of the DIRS framework is compared with the following benchmark methods for real-time decisions.

1) Random offloading (Random) denotes that the offloading decision is decided randomly for each IoTD. If the computational resource of the allocated MEC is insufficient, IoTD executes the task locally.
2) All offloading (Remote) denotes that all the IoTDs offload the tasks to the nearest MEC. If the computational resource is insufficient, the IoTDs who need more computing resources execute the task locally.
3) Local execution (Local) denotes that all IoTDs decides to execute the task locally.
4) MADDPG is a celebrated multiagent DDPG.

In Fig. 7, we compare the sum of energy and time consumption for all UEs (Objective function) between the proposed framework, MADDPG, all offloading, random offloading, and local execution. As shown in Fig. 7, the proposed method achieves the lowest value of the objective function when the number of IoTDs is large, which implies that the DIRS framework is suitable for the large-scale IoT scenario. Moreover, one can see that the MADDPG also obtains better performance than remote, random, and local strategies.

## VI. CONCLUSION

In this article, a novel DIRS framework has been proposed for large-scale MEC systems. This framework adopts a distributed DRL to jointly optimize computation offloading, transmission power, and recourse allocation, with the objective of minimizing the sum of task latency and energy consumption for all the IoTDs. Overall, the proposed DIRS framework has the following three advantages.

1) The proposed DIRS framework consists of a multiagent ensemble assisted DRL architecture for centralized learning and distributed inference.
2) A Lévy flight search is proposed as the action refinement module of DRL for more efficient exploration in large action space.

3) An imitation acceleration is presented to help agents to achieve better performance in complex environment by pretraining with the demonstration data.

The simulation results demonstrate that the DIRS framework has better performance than the existing benchmarks, and it exhibits enormous potential in the large-scale application scenarios.

In the future, we plan to apply our proposed framework to more complex environment, for instance, in the scenario where UAV is served as the edge computing node [3]. Additionally, we plan to propose the prediction model to predict the moving pattern of the mobile users based on the historical data. Then, we can integrate the prediction result to our proposed solution to further improve the scalability of the framework.

## REFERENCES


[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[2] S. Yu, B. Dab, Z. Movahedi, R. Langar, and L. Wang, "A socially-aware hybrid computation offloading framework for multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1247–1259, Jun. 2020.

[3] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "Deep-learning-based joint resource scheduling algorithms for hybrid mec networks," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6252–6265, Jul. 2020.

[4] L. Zhang *et al.*, "LNTP: An end-to-end online prediction model for network traffic," *IEEE Netw.*, vol. 35, no. 1, pp. 226–233, Jan./Feb. 2021.

[5] D. Liu, L. Khoukhi, and A. Hafid, "Decentralized data offloading for mobile cloud computing based on game theory," in *Proc. IEEE 2nd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, 2017, pp. 20–24.

[6] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Trans. Comput.*, vol. C-26, no. 9, pp. 917–922, Sep. 1977.

[7] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 1. Belmont, MA, USA: Athena Sci., 1995.

[8] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[9] 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.

[10] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "AI driven heterogeneous MEC system with UAV assistance for dynamic environment: Challenges and solutions," *IEEE Netw.*, vol. 35, no. 1, pp. 400–408, Mar./Apr. 2021.

[11] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[12] Y.-J. Liu, S.-M. Cheng, and Y.-L. Hsueh, "eNB selection for machine type communications using reinforcement learning based Markov decision process," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 11330–11338, Dec. 2017.

[13] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[14] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[15] 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.

[16] C. H. Liu, Z. Chen, J. Tang, J. Xu, and C. Piao, "Energy-efficient UAV control for effective and fair communication coverage: A deep reinforcement learning approach," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 9, pp. 2059–2070, Sep. 2018.

[17] M. Chen, T. Wang, K. Ota, M. Dong, M. Zhao, and A. Liu, "Intelligent resource allocation management for vehicles network: An A3C learning approach," *Comput. Commun.*, vol. 151, pp. 485–494, Feb. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419314215

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[19] H. Li, K. Ota, and M. Dong, "Deep reinforcement scheduling for mobile crowdsensing in fog computing," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–18, Apr. 2019. [Online]. Available: https://doi.org/10.1145/3234463

[20] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultra-dense network," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2238–2251, Feb. 2021.

[21] H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep reinforcement learning based resource allocation for V2V communications," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3163–3173, Apr. 2019.

[22] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," 2018. [Online]. Available: arXiv:1812.07394.

[23] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed multi-agent meta learning for trajectory design in wireless drone networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 10, pp. 3177–3192, Oct. 2021.

[24] M. Chen *et al.*, "Distributed learning in wireless networks: Recent progress and future challenges," 2021. [Online]. Available: arXiv:2104.02151.

[25] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[26] J. Du *et al.*, "Resource pricing and allocation in MEC enabled blockchain systems: An A3C deep reinforcement learning approach," *IEEE Trans. Netw. Sci. Eng.*, early access, Mar. 24, 2021, doi: 10.1109/TNSE.2021.3068340.

[27] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, Aug. 2019.

[28] Y. Hua, R. Li, Z. Zhao, X. Chen, and H. Zhang, "GAN-powered deep distributional reinforcement learning for resource management in network slicing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 334–349, Feb. 2020.

[29] K. Wang, K. Yang, and C. S. Magurawalage, "Joint energy minimization and resource allocation in C-RAN with mobile cloud," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 760–770, Sep. 2018.

[30] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, 2013.

[31] W. Yang, P. Dong, L. Cai, and W. Tang, "Loss-aware throughput estimation scheduler for multi-path TCP in heterogeneous wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 5, pp. 3336–3349, May 2021.

[32] 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.

[33] J. Wallenius, J. S. Dyer, P. C. Fishburn, R. E. Steuer, S. Zionts, and K. Deb, "Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead," *Manag. Sci.*, vol. 54, no. 7, pp. 1336–1349, 2008. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.1070.0838

[34] G. Dulac-Arnold *et al.*, "Deep reinforcement learning in large discrete action spaces," 2015. [Online]. Available: arXiv:1512.07679.

[35] T. Hester *et al.*, "Deep Q-learning from demonstrations," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3223–3230.

[36] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 92–99, Feb. 2020.

[37] Y. Liu, W. Zhang, S. Pan, Y. Li, and Y. Chen, "Analyzing the robotic behavior in a smart city with deep enforcement and imitation learning using IoRT," *Comput. Commun.*, vol. 150, pp. 346–356, Mar. 2020.

[38] E. Emary, H. M. Zawbaa, and M. Sharawi, "Impact of Lèvy flight on modern meta-heuristic optimizers," *Appl. Soft Comput.*, vol. 75, pp. 775–789, Feb. 2019.
[39] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, Sep. 2017.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980.
[41] L. Huang, S. Bi, and Y. J. 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.
[42] F. Jiang, L. Dong, and Q. Dai, "Designing a mixed multilayer wavelet neural network for solving ERI inversion problem with massive amounts of data: A hybrid STGWO-GD learning approach," *IEEE Trans. Cybern.*, early access, May 20, 2020, doi: 10.1109/TCYB.2020.2990319.
[43] F. Jiang, K. Wang, L. Dong, C. Pan, and K. Yang, "Stacked autoencoder-based deep reinforcement learning for online resource scheduling in large-scale MEC networks," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9278–9290, Oct. 2020.
[44] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2651–2664, Dec. 2018.
[45] Y. Wang, Z.-Y. Ru, K. Wang, and P.-Q. Huang, "Joint deployment and task scheduling optimization for large-scale mobile users in multi-UAV-enabled mobile edge computing," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3984–3997, Sep. 2020.

**Feibo Jiang** (Member, IEEE) received the B.S. and M.S. degrees from the School of Physics and Electronics, Hunan Normal University, Changsha, China, in 2004 and 2007, respectively, and the Ph.D. degree from the School of Geosciences and Info-Physics, Central South University, Changsha, in 2014.

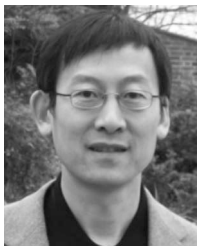
He is currently an Associate Professor with the Hunan Provincial Key Laboratory of Intelligent Computing and Language Information Processing, Hunan Normal University. His research interests include artificial intelligence, fuzzy computation, Internet of Things, and mobile edge computing.

**Li Dong** received the B.S. and M.S. degrees from the School of Physics and Electronics, Hunan Normal University, Changsha, China, in 2004 and 2007, respectively, and the Ph.D. degree from the School of Geosciences and Info-Physics, Central South University, Changsha, in 2018.

Her research interests include machine learning, Internet of Things, and mobile edge computing.

**Kezhi Wang** (Senior Member, IEEE) received the B.E. and M.E. degrees from the School of Automation, Chongqing University, Chongqing, China, in 2008 and 2011, respectively, and the Ph.D. degree in engineering from the University of Warwick, Coventry, U.K., in 2015.

He was a Senior Research Officer with the University of Essex, Colchester, U.K. He is currently a Senior Lecturer with the Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, U.K. His research interests include wireless communication, mobile edge computing, UAV communication, and machine learning.



**Kun Yang** (Senior Member, IEEE) received the Ph.D. degree from the Department of Electronic and Electrical Engineering, University College London (UCL), London, U.K., in 2006.

He is currently a Chair Professor with the School of Computer Science and Electronic Engineering, University of Essex, Colchester, U.K., where he leading the Network Convergence Laboratory. He is also an Affiliated Professor with University of Electronic Science and Technology of China, Chengdu, China. Before joining in the University of Essex in 2003, he worked with UCL on several European Union (EU) research projects for several years. He manages research projects funded by various sources, such as U.K. EPSRC, EU FP7/H2020, and industries. He has published over 150 journal papers and filed ten patents. His main research interests include wireless networks and communications, IoT networking, data and energy integrated networks, and mobile edge computing.

Dr. Yang serves on the editorial boards of IEEE and non-IEEE journals. He has been a Fellow of IET since 2009.

**Cunhua Pan** (Member, IEEE) received the B.S. and Ph.D. degrees from the School of Information Science and Engineering, Southeast University, Nanjing, China, in 2010 and 2015, respectively.

From 2015 to 2016, he was a Research Associate with the University of Kent, Canterbury, U.K. He held a postdoctoral position with the Queen Mary University of London, London, U.K., from 2016 and 2019, where he is currently a Lecturer. His research interests mainly include reconfigurable intelligent surfaces (RIS), intelligent reflection surface, ultra-reliable low latency communication, machine learning, UAV, Internet of Things, and mobile edge computing.

Dr. Pan serves as a TPC member for numerous conferences, such as ICC and GLOBECOM, and the Student Travel Grant Chair for ICC 2019. He is currently an Editor of IEEE WIRELESS COMMUNICATION LETTERS, IEEE COMMUNICATIONS LETTERS, and IEEE ACCESS. He also serves as a Leading Guest Editor of IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING Special Issue on Advanced Signal Processing for Reconfigurable Intelligent Surface-aided 6G Networks, *IEEE Vehicular Technology Magazine* on the special issue on Backscatter and Reconfigurable Intelligent Surface Empowered Wireless Communications in 6G, IEEE OPEN JOURNAL OF VEHICULAR TECHNOLOGY on the special issue of Reconfigurable Intelligent Surface Empowered Wireless Communications in 6G and Beyond, and the IEEE ACCESS Special Issue on Reconfigurable Intelligent Surface Aided Communications for 6G and Beyond. He is Workshop Organizer in IEEE ICCC 2021 on the topic of Reconfigurable Intelligent Surfaces for Next Generation Wireless Communications (RIS for 6G Networks), and Workshop Organizer in IEEE Globecom 2021 on the topic of Reconfigurable Intelligent Surfaces for future wireless communications. He is currently the Workshops and Symposia officer for Reconfigurable Intelligent Surfaces Emerging Technology Initiative.