Multi-Agent Imitation Learning for Pervasive Edge Computing: A Decentralized Computation Offloading Algorithm

Xiaojie Wang, *Member, IEEE*, Zhaolong Ning, *Senior Member, IEEE*,
and Song Guo, *Fellow, IEEE*

Abstract—Pervasive edge computing refers to one kind of edge computing that merely relies on edge devices with sensing, storage and communication abilities to realize peer-to-peer offloading without centralized management. Due to lack of unified coordination, users always pursue profits by maximizing their own utilities. However, on one hand, users may not make appropriate scheduling decisions based on their local observations. On the other hand, how to guarantee the fairness among different edge devices in the fully decentralized environment is rather challenging. To solve the above issues, we propose a decentrailized computation offloading algorithm with the purpose of minimizing average task completion time in the pervasive edge computing networks. We first derive a Nash equilibrium among devices by stochastic game theories based on the full observations of system states. After that, we design a traffic offloading algorithm based on partial observations by integrating general adversarial imitation learning. Multiple experts can provide demonstrations, so that devices can mimic the behaviors of corresponding experts by minimizing the gaps between the distributions of their observation-action pairs. At last, theoretical and performance results show that our solution has a significant advantage compared with other representative algorithms.

Index Terms—Pervasive edge computing, computation offloading, imitation learning, decentralized execution

1 Introduction

Edge computing extends the traditional cloud computing architecture by leveraging computation and storage resources on the network edge [1], [2]. Clouds can schedule tasks to be processed locally by edge devices without remote transmission. With the development of 5G and networking technologies, terminal devices have been evolved with powerful sensing, computing and storage capacities, which paves the way to realize pervasive edge computing [3]. Actually, it is a novel kind of edge computing, which merely leverages edge devices for computation and storage without centralized management. Traditional edge computing is a supplementation of cloud computing, where computation and storage resources are provided by edge servers, and decisions are made in the backend [4]. Contrastively, pervasive edge computing allows data storage, processing and scheduling decisions to be all performed at the network edge. Thus, traditional edge computing strategies are not suitable for the pervasive edge computing environment, calling for novel algorithm designs in a fully decentralized manner.

The advantages brought by pervasive edge computing compared with traditional edge computing can be summarized into four aspects. First, it is infrastructure-free for deploying and maintaining the dedicated cloud backend. Second, there is no need to communicate with the cloud, since data can be processed near users, which greatly reduces the transmission delay. Furthermore, it is connectivity-independent by fulfilling communications among peer devices without the requirement of Internet connectivities. At last, no central authority is required, and devices are free to make decisions about how to collaborate with others and in what way to enable feasible and diverse network applications. The applications of pervasive edge computing are widely spread from entertainment to industry. For example, on the scene of a live basketball game, audiences sitting on different locations can share their recorded videos from their angles with others through peer-to-peer communications [5]. Then, a multi-angle viewing game video can be formed by aggregating different fragments. Audiences at different locations can have a panoramic view over the live game. Another example is cooperative driving, where live video streaming of road conditions and accident scenes can be shared directly among vehicles based on short-distance communication technologies [6].

Although pervasive edge computing can bring various advantages and conveniences for users, it is challenging to

- *Xiaojie Wang is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China, and also with the Chongqing Key Laboratory of Mobile Communications Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. E-mail: xiaojie.wang@polyu.edu.hk.*
- *Zhaolong Ning is with the School of Software, Dalian University of Technology, Dalian, Liaoning 116044, China, and also with the Chongqing Key Laboratory of Mobile Communications Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China. E-mail: zhaolongning@dlut.edu.cn.*
- *Song Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: song.guo@polyu.edu.hk.*

*Manuscript received 19 Apr. 2020; revised 27 July 2020; accepted 9 Sept. 2020. Date of publication 15 Sept. 2020; date of current version 25 Sept. 2020.
(Corresponding authors: Zhaolong Ning; Song Guo.)
Recommended for acceptance by Y. Yang.
Digital Object Identifier no. 10.1109/TPDS.2020.3023936*

design a feasible computation offloading algorithm by considering the fairness on utilities of multiple devices in the pervasive edge computing networks [7]. The challenges can be summarized as follows:

- Compared with traditional edge computing, pervasive edge computing allows devices to make decisions on the network edge without centralized management. Devices are difficult to obtain the entire network status by merely depending on peer-to-peer communications. Thus, it is challenging for them to select proper edge servers (formed by other devices) to offload their tasks based on partial observations. Impacted by that, the task completion time is difficult to guarantee without reasonable task allocation strategies.
- In the multi-device environment, each device intends to maximize its own utility [8]. Existing researches always develop game theoretical models to compute a Nash equilibrium [9]. For each device, it bargains with others based on the global knowledge of the system state. However, in the pervasive edge computing network, devices cannot obtain the global information, thus how to ensure the fairness of devices in a fully decentralized environment deserves to be investigated.
- Under partial observations, it is applicable to design learning-based methods that can obtain good policies through interacting with environments. However, on one hand, the performance of existing model-free learning methods at the beginning stage is always poor, and they are not suitable for online scheduling. On the other hand, their convergence speeds are slow, especially in the partial-observable environment with multiple agents. Thus, it is necessary to design a learning method that has a fast convergence speed and can be executed in a decentralized manner.

To solve the above challenges, this paper proposes a multi-agent imitation learning based computation offloading algorithm for pervasive edge computing, named MILP, with the purpose of minimizing the average task completion time of devices. They can either offload tasks to other devices for computation, or process them locally, totally depending on their own observations and decisions. To the best of our knowledge, this paper is an early effort to investigate the computation offloading issue for pervasive edge computing based on multi-agent imitation learning. Generally, imitation learning is a kind of machine learning methods, and it allows the learning agent to imitate expert policies, which can effectively solve the original problem but cannot be conducted in an online manner, due to its high time complexity. Hence, a training process is designed to learn agent policies by realizing imitation from the expert. Furthermore, multi-agent imitation learning allows multiple agents to imitate the behaviors of corresponding experts, and can reach a Nash equilibrium among agents. Specially, our contributions can be summarized as follows:

- We formulate the task scheduling issue in the pervasive edge computing environment as an optimization problem by considering communication and computation abilities of edge devices. To resolve it, we establish the relationship between the original optimization problem and stochastic games by specifying game elements, such as evolved players, states and state transition possibilities, and transform the optimization problem into a reward maximization problem.
- To resolve the reward maximization problem, we relax the constraints brought by pervasive edge computing networks, and propose a multi-agent imitation learning based computation offloading algorithm, which allows multiple learning agents to imitate the behaviors of corresponding experts for good scheduling policies. *To the best of our knowledge, this is the first work to integrate multi-agent Generalized Adversarial Imitation Learning (GAIL) with pervasive edge computing to solve the traffic scheduling issue*.
- To form the expert policies, we adopt Actor-Critic with Kronecker-factored Trust Region (ACKTR) algorithm to find the optimal strategies of experts based on the full observations of system states. For the agent policies, a novel neural network model for each device based on partial observations is presented. It comprehensively integrates Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs) and ACKTR to approach the expert performance, and can be executed in an online manner.
- We demonstrate the superiority of our algorithm from both theoretical and experimental perspectives. Theoretical results demonstrate that the algorithm can guarantee the fairness of devices and reach Nash equilibrium based on both full and partial observations. Performance results show that our algorithm has advantages on average task completion time, convergence time and offloading ratios.

The rest of this paper is structured as follows: in Section 2, we review the related work; we present the pervasive edge computing model and formulate the studied problem in Section 3; in Section 4, we design a multi-agent imitation learning based computation offloading algorithm, followed by performance evaluation in Section 5; finally, we conclude our work in Section 6.

2 Related Work

In this section, we review the state-of-the-art researches about pervasive edge computing and imitation learning.

2.1 Pervasive Edge Computing

Recent studies on pervasive edge computing focus on how to improve resource utilization efficiency and satisfy user requirements by designing reasonable resource management strategies. A distributed computational load balancing algorithm is proposed in [10] to improve the overall network performance of a battery-powered sensor system. The authors in [11] study the caching fairness problem in the pervasive edge computing environment, where a distributed caching node selection algorithm is designed to realize load balancing among different edge devices. The blockchain technology is adopted in [12] to allocate storage resources on edge devices. The undeniable and unmodifiable transactions can be guaranteed by a distributed resource allocation algorithm. Although these algorithms investigate the resource allocation problem by considering the fairness among different edge devices, they neglect the intends of edge devices that prepare to maximize their own utilities in the decentralized environment.

In addition, some researches prepare to improve system performance by minimizing the task execution time. A decentralized algorithm is proposed in [13] to schedule computational tasks among the edge cloud and nearby devices with the purpose of improving their network performance. Game theoretical model is established, and a Nash equilibrium is reached based on the average system parameters. The authors in [14] intend to minimize task execution delay by leveraging pervasive grids, which are composed by under-explored and idle resources in sensing devices and instruments. A distributed computation platform is designed to ensure context awareness and data proximity. The authors in [15] investigate the task offloading issue for low-latency application requirements in a distributed manner. Quality-of-Experience (QoE) of users is considered in the pervasive edge computing environment [16], with the purpose of improving the data accuracy and the transmission rate. A conventional neural network based algorithm is proposed to schedule tasks among different devices. To minimize the transmission overhead, the authors in [17] propose a peer data sharing algorithm that enables devices to discover neighbors with cached data. Although these researches have similar purposes with this paper, their solutions either depend on a centralized management server or relay on the full knowledge of system states, and cannot be executed in a fully decentralized manner.

2.2 Imitation Learning

Imitation learning is a kind of machine learning that enables the learning agent to mimic the behaviors of experts for good performance. Generally, expert policies can reach optimal performance in the network. However, they cannot be applied directly since the complexity or dependent system parameters are difficult to obtain based on current network conditions [18]. The authors in [19] design an efficient learning algorithm that generates policies iteratively by feedbacking information to the learning agent based on the demonstrations of experts. To reduce the compound errors caused by this supervised manner, GAIL is proposed in [20]. It derives a model-free learning algorithm that outperforms other imitation learning methods, by fitting distributions of state-action pairs related to expert behaviors.

To realize the interactions among multiple agents, a multi-agent GAIL algorithm is proposed in [21], which allows learning agents to imitate complex expert behaviors in a high-dimensional environment. The authors in [22] model the interactions among multiple agents based on imitation learning by considering correlated policies. A Nash equilibrium needs to be reached among different agents, since user fairness and system stability should be guaranteed in the multi-agent environment. Due to its high-efficiency and fast convergence speed, we integrate multi-agent imitation learning with our system to minimize the task completion time. To the best of our knowledge, we are the first to apply multi-agent imitation learning in pervasive edge computing networks. Nevertheless, it cannot be directly applied in our considered task scheduling problem, since the fully-decentralized environment and complicated network conditions bring great challenges on the application of multi-agent imitation learning.

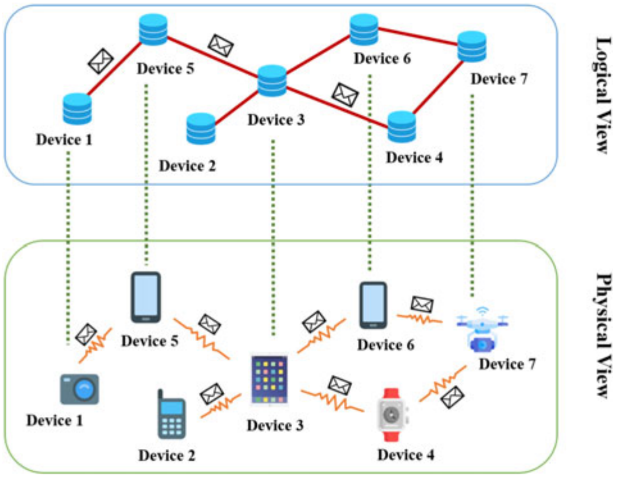


Fig. 1. An illustrative system model.

3 System Model And Problem Formulation

In this section, we first overview the designed system, and then describe the communication and computation models. After that, the problem is formulated.

3.1 System Overview

As shown in Fig. 1, we consider a wireless network consisting of multiple devices, denoted by $\mathcal{N} = \{1, \ldots, i, \ldots, N\}$. In time slot t, device $i \in \mathcal{N}$ generates a set of tasks represented by $X_i^t = \left\{x_{i,1}, \ldots, x_{i,K(i,t)}\right\}$, where $K(i,t)$ is the total number of tasks device i can generate in time slot t. The size of task $x_{i,k}$ is defined by $s_{i,k}$, and its required CPU cycle for computation is $c_{i,k}$. Similar to [13] and [23], the task generation process of each device can be modeled as a Poisson process with generation intensity Λ_i^t. For task $x_{i,k}$, device i can either offload it to device $j \in \mathcal{N} \backslash \{i\}$ or process it locally. The flag of assigning task $x_{i,k}$ to device j can be represented by binary value $f_{i,k,j}$, and $\sum_{j=1}^{N} f_{i,k,j} = 1$. We denote the flag vector for assigning task $x_{i,k}$ by $f_{i,k} = \left\{f_{i,k,1}, \ldots, f_{i,k,N}\right\}$. The main notations of this paper is illustrated in Table 1.

There is no centralized authority in our considered system, and each device maintains a status list locally. When a device first joins in the network, the list only contains the status about its own transmission and processing queues, current speed, location, and moving direction. At the beginning of each time slot, devices broadcast all the records in their status lists to neighboring ones. Then, devices update their local status lists based on the received records. For example, there are four relative static nodes in the network and each one maintains a status list locally in Fig. 2. Initially, each node only knows its own status. As time goes by, nodes can learn the status of directly connected ones recorded in the current time slot, while learning those of indirect ones recorded in previous time slots.

For instance, node A in time slot 1 merely has its own status, and it receives the status record of node C since they have a direct connection in time slot 2. In time slot 3, node A learns the status of other indirect-connected ones from the records in node C updated in time slot 2, and receives the updated status of node C in time slot 3. Herein, we utilize ξ_{ij}^t to represent the connection status of devices i and j in

TABLE 1
Main Notations

Notation	Description
$\mathcal{N}$	The group of devices in the network.
N	The total number of devices.
X_i^t	The group of tasks generated by each device in time slot t.
$K(i,t)$	The total number of tasks that device i can generate in time slot t.
$s_{i,k}$	The size of task $x_{i,k}$.
$c_{i,k}$	The required CPU for task $x_{i,k}$.
Λ_i^t	The task generation intensity of device i in time slot t.
ξ_{ij}^t	The connection status of devices i and j in time slot t.
Υ_{ij}	The transmission rate from devices i to j.
$T(i)$	The average task completion time for device i.
s^t	The system state in time slot t.
o_i^t	The local observation of client i in time slot t.
a_i^t	The action that device i can take in time slot t.
a_{ik}^t	The action that device i can take for task $x_{i,k}$ in time slot t.
$p(s^{t+1} \mid s^t, \boldsymbol{a}^t)$	The state transition possibility from s^t to s^{t+1} according to joint action $\boldsymbol{a}^t$.
$r(s^t, a_i^t)$	The reward received by device i in time slot t based on state s^t and action a_i^t.
$\pi_i(a_{ik}^t \vert s^t)$	The possibility of taking action a_{ik}^t by policy π_i at state s^t.
$v_i(s^t)$	The state value of device i at state s^t.
$q_i(s^t, a_i^t)$	The value-action function of device i related to state s^t and action a_i^t.
$\nabla_{\theta_i} L^E(\theta_i)$	The gradient of policy network for expert i.
$L^E(\phi_i)$	The loss function of value network for expert i.

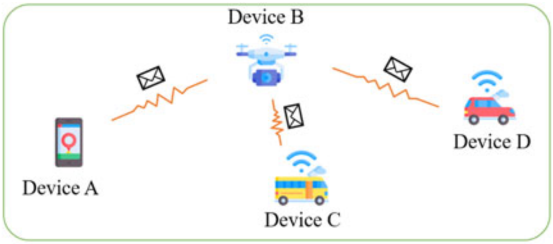
time slot t. If $\xi_{ij}^t = 1$, devices i and j can directly or indirectly establish a link with each other in time slot t. That is, they can connect with each other while in neighborhood, or fulfill transmissions via a set of relay nodes. In addition, we assume that devices can faithfully process tasks for others, guaranteed by existing security and incentive mechanisms. Meanwhile, the main concern of this paper is to realize online task scheduling, thus we do not consider link disconnection happens during task processing.

3.2 Communication and Computation Models

We consider that devices can communicate with each other through Orthogonal Frequency-Division Multiple Access (OFDMA) technology, which enables the subcarrier of each device to be orthogonal to others. Thus, total $(N-1) \times (N-1)$ subcarriers are needed for the communications among devices in $\mathcal{N}$. Each device maintains $N-1$ transmission queues locally for transmitting tasks to other devices. When devices i and j are in neighborhood, the transmission delay for task $x_{i,k}$ can be computed by $T_d(i,k,j) = s_{i,k}/\Upsilon_{ij}$, where Υ_{ij} is the transmission rate from devices i to j. The tasks are transmitted following a First-In, First-Out (FIFO) order. Then, the waiting time for task $x_{i,k}$ in the transmission queue from devices i to j can be obtained based on the $M/G/1$ queueing system [24]:

$$T_w(i,k,j) = \frac{\left(\mu_{ij}^t \bar{d}_{i,j} + \mu_{ij}^t \delta^2\right)}{2(1-\mu_{ij}^t \bar{d}_{i,j})}, \tag{1}$$

where $\bar{d}_{i,j}$ is the average transmission delay for tasks from devices i to j; symbol δ^2 is the variance for transmission delay; and μ_{ij}^t reflects the task transmission intensity from devices i to j.



Devices	Status Lists		
	t=1	t=2	t=3
A	s_A^1	s_A^2, s_C^2	$s_A^3, s_C^3, s_B^2, s_D^2$
B	s_B^1	s_B^2, s_C^2	$s_B^3, s_C^2, s_A^2, s_D^2$
C	s_C^1	$s_C^2, s_A^2, s_B^2, s_D^2$	$s_C^3, s_A^3, s_B^3, s_D^3$
D	s_D^1	s_D^2, s_C^2	$s_D^3, s_A^2, s_B^2, s_C^3$

Fig. 2. An example of status list update.

For task offloading, we consider that tasks from device i can be offloaded to devices several hops away. That is to say, the tasks from device i can be transmitted through several relay nodes to the destination device for processing. Thus, when a connection path is established between devices i and j through several relay nodes, the total transmission delay of $x_{i,k}$ from devices i to j can be computed by:

$$T_d(i,k,j) = \sum_{l \in \mathcal{N} \setminus \{j\}} [T_d(i,k,l) + T_w(i,k,l)]\beta_{i,k,l}, \tag{2}$$

where $\beta_{i,k,l}$ is a binary value denoting whether device l is a relay node that helps to transmit task $x_{i,k}$ to device j. In addition, we consider that $T_d(i,k,i) = 0$.

After task $x_{i,k}$ is received by device j, it first waits in the processing queue, and then is served according to the FIFO order. Waiting delay $T_q(i,k,j)$ in the processing queue can be obtained similar to equation (1). The processing delay consumed for task $x_{i,k}$ by device j is $T_p(i,k,j) = c_{i,k}/b_j$, where b_j is the computation ability of device j, i.e., CPU cycles per second. Similar to [13] and [23], we consider that the transmission delay for the computation result can be neglected, since its size is small enough.

3.3 Problem Formulation

In time slot t, when device i has tasks to compute, it can either offload them to other devices or process them locally. For task $x_{i,k}$, the average task execution delay can be computed by:

$$T(i,k) = \sum_{j \in \mathcal{N}} f_{i,k,j}\big(T_d(i,k,j) + T_q(i,k,j) + T_p(i,k,j)\big). \tag{3}$$

Then, the average task completion time for device i is:

$$T(i) = \lim_{\mathbb{T} \to \infty} \frac{1}{\mathbb{T}} \sum_{t=1}^{\mathbb{T}} \frac{1}{K(i,t)} \sum_{k=1}^{K(i,t)} T(i,k), \tag{4}$$

where $\mathbb{T}$ is the total time slots that the algorithm has been running. The purpose of each device is to minimize its average task completion time, i.e.,

$$\text{P1:} \quad \min_{j,\beta_{i,k,l}} T(i), \quad j,l \in \mathcal{N}, \tag{5}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} f_{i,k,j}\xi_{ij}^{t} = 1. \tag{5a}$$

Constraint (5a) ensures task $x_{i,k}$ should be assigned to the device that can establish a connection directly or through several relay nodes with device i. Since the task scheduling decision of one device can be affected by others, a bargain process should be conducted to achieve the fairness among different devices. Typically, a non-cooperative game can be leveraged to solve the above issue [13], where the system state should be known by all devices for decision making. However, in our considered decentralized environment, devices cannot obtain the instantaneous system state due to the fully-decentralized environment. Thus, we utilize multi-agent imitation learning in our system, and each agent does not need to be aware of the full system state but trains policies by following experts' demonstrations based on its local observations. In the next section, we specify our designed MILP algorithm.

4 A Multi-Agent Imitation Learning Based Computation Offloading Algorithm

In this section, we present the designed multi-agent imitation learning based computation offloading algorithm. We first provide the algorithm overview, and then describe it in detail.

4.1 Algorithm Overview

To solve the formulated problem, the designed multi-agent imitation-learning based computation offloading algorithm can be divided into the following steps:

1) *Problem Transformation*: Since Problem P1 cannot be directly solved in such a decentralized environment, we first establish the relationship between the formulated optimization problem and the stochastic game. Then, we specify the game elements by defining states, observations, actions and transition possibilities related to our considered scenario. After that, the delay minimization problem in P1 can be transmitted into a reward-maximization problem in P2. In our considered game, we not only consider the actions taken by each device in each time slot, but also need to choose the action that should be taken for each task. Thus, we formulate the unique value function and Nash equilibrium conditions in our considered system. The reward-maximization problem can be further transferred into a Lagrangian dual problem, which can be solved in the decentralized environment. The detailed description for problem transformation is in Sections 4.2 and 4.3.
2) *Expert policy acquisition*: In imitation learning, expert policies are important factors to affect the final performance of agent policies. Thus, we should design an efficient algorithm to derive the expert demonstrations. We consider that the experts can observe the full system states, and they can solve the dual problem by a natural gradient policy, such as ACKTR [25], in an offline manner. Consequently, a Nash equilibrium can be achieved among devices, and expert demonstrations can be formed by collecting their observation-action pairs, as specified in Section 4.4.
3) *Agent policy acquisition*: In the considered pervasive edge computing networks, only local observations are available to each device. To approach the performance of expert policies based on full system states, we design a novel neural network model by integrating CNNs, GANs [26] and ACKTR, which can be conducted in an online manner to minimize the gaps between observation-action distributions of the corresponding experts and agents. The design of agent polices is presented in Section 4.5.
4) *Task scheduling*: By agent policies, each device can get action a_{ik}^{t} for its local task x_{ik}, i.e., on which device to compute task x_{ik}. Then, task x_{ik} can be transmitted from devices i to k. For the transmission, a direct or indirect path through several relay nodes can be established. Since the routing issue in the decentralized wireless networks has been well investigated such as [27] and [28], we do not specify it here.

4.2 Stochastic Game Formulation

Since there are multiple devices in our considered system, they can either offload their tasks to other devices when they are overloaded, or help others to compute the offloaded tasks when they are underloaded. In order to reach a Nash equilibrium among devices in $\mathcal{N}$, their interactions can be modeled as a stochastic game, which can be represented by a tuple $\langle \mathcal{N}, S, O, A, P, R, \rho_0, \gamma \rangle$. In the following, we describe how to set the elements in the tuple based on the interactions of devices in the pervasive edge computing environment.

1) S represents the state space for our decentralized system, containing all possible states related to devices and tasks. We utilize s^t to denote the system state in time slot t, where $s^t = \{s_g^t, s_a^t, s_p^t, s_b^t, s_i^t\}$. Symbol s_g^t is the instantaneous network connected graph in time slot t; Symbols s_a^t and s_p^t represent the status of transmission and processing queues of each device, respectively; the processing ability of the device is denoted by s_b^t, while the task generation intensity is s_i^t.
2) O is the observation space for devices in the system. Since there is not a centralized coordination in our decentralized system, the observations of devices are different from the system states. We utilize o_i^t to denote the observation of device i in time slot t, and $o_i^t \neq s^t$. Herein, o_i^t can include part of the system state recorded in previous time slots while others in the current one.
3) A is the action space for tasks. In our considered system, we not only consider the actions of devices, but also choose actions for each task. From the perspective of devices, vector $\boldsymbol{a}^t$ is the group of actions for all devices in time slot t, denoted by $\{a_1^t, a_2^t, \ldots, a_N^t\}$. Action a_i^t represents the action that device i can take in time slot t, i.e., whether offloading its local tasks or not, and which device should be chosen to process its offloaded tasks. From the perspective of tasks, we

utilize a_{ik}^t to denote the action for task $x_{i,k}$, since there can be more than one task on device i need to be scheduled in time slot t. Thus, $a_i^t = \{a_{i1}^t, a_{i2}^t, \ldots a_{ik}^t, \ldots\}$. When the task is processed locally, action $a_{ik}^t = 0$; otherwise $a_{ik}^t = j$. Thus, the action space can be represented by $A = \{1, 2, \ldots, j, \ldots, N\}$. We assume that each device decides its action dependently and does not know the actions of others.

4) $\mathcal{P}$ denotes the state transition possibility matrix, where $p(s^{t+1} \mid s^t, \boldsymbol{a}^t)$ denotes the state transition possibility from s^t to s^{t+1} according to joint action $\boldsymbol{a}^t$.
5) R represents the reward group, where $r(s^t, a_i^t)$ is the reward received by device i in time slot t based on system state s^t and chosen action a_i^t. By taking action a_{ik}^t, the reward for one task can be computed by $r(s^t, a_{ik}^t) = -T(i,k)$.
6) Symbol ρ_0 is the distribution of initial system state s^0, and $\gamma \in (0,1)$ is the discounted factor of the expected reward.

Based on the above formulated stochastic game, we can establish the relationship between the average task completion time and the cumulative reward as follows:

Theorem 1 (The Cumulative Reward). *In our considered system, the average task completion time of each device can also be redescribed as* $T(i) \doteq -E\big[\sum_{\tau=0}^{\infty} \gamma^\tau r(s^{t+\tau}, a_i^{t+\tau})\big]$*, where operator "$\doteq$" means "approach to".*

The proof can be found in Appendix A of Supplementary File, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2020.3023936.

Based on Theorem 1, the purpose of our algorithm becomes to maximize the expected cumulative discounted reward as follows:

$$\text{P2:} \quad \max_{\boldsymbol{a}} E\left[\sum_{\tau=0}^{\infty} \gamma^\tau r(s^{t+\tau}, a_i^{t+\tau})\right], \tag{6}$$

$$\text{s.t.} \quad \text{Constraint (5a).}$$

Then, each device tries to maximize its cumulative discounted reward in the considered pervasive edge computing network. For multiple devices, they should design a neural network that can be executed in a decentralized manner and guarantee a Nash equilibrium among them. We utilize π_i and π_{-i} to denote the strategies taken by device i and other devices, respectively. Meanwhile, policies π_{-i}^* and π_i^* represent the corresponding optimal policies. To reach the Nash equilibrium in our considered system, the following conditions should be satisfied:

Definition 1 (Nash equilibrium). *In our considered system, when Markov strategy* $\pi^* = \langle \pi_1^*, \pi_2^*, \ldots, \pi_N^* \rangle$ *trained by devices satisfies* $v_{\pi_i^*, \pi_{-i}^*}(s^t) \geq v_{\pi_i, \pi_{-i}^*}(s^t)$*, where* $i \in \mathcal{N}$ *and* $s^t \in S$*, the formulated stochastic game in Section 4.2 can reach a Nash equilibrium, where* $v_{\pi_i, \pi_{-i}}(s^t)$ *is the state value of device* i *and can be computed by* $v_{\pi_i, \pi_{-i}}(s^t) = E\big[\sum_{\tau=0}^{\infty} \gamma^\tau r(s^{t+\tau}, a_i^{t+\tau}) | s^t = s^0\big]$.

In the following, we mention variables $v_{\pi_i, \pi_{-i}}(s^t)$ and $v_i(s^t)$ interchangeably.

4.3 Optimization Problem Transformation

Although we have transferred the original optimization problem P1 into an equivalent problem P2, it is still difficult to solve P2. The reasons are as follows: 1) The constraint of problem P2 makes it difficult to meet when a neural network is being trained; 2) The Nash equilibrium in Definition 1 requires the information of policies generated by other devices, which cannot be obtained in the fully-decentralized environment. Thus, problem P2 should be further transferred into a solvable form.

We first process the constraint in P2. According to the formulated stochastic game, we can obtain policy π_i that maps state s^t to corresponding action a_{ik}^t by computing probabilities for choosing each action, thus $\sum_{a_{ik}^t \in A} \pi_i\big(a_{ik}^t | s^t\big) = 1$. From constraint (5a), we can know that the task must be offloaded to the device that can establish a directly or indirectly connection with its owner. Correspondingly, the output of the neural network should ensure that $\sum_{a_{ik}^t \in A} \pi_i\big(a_{ik}^t | s^t\big)\xi_{ij}^t = 1, j \in A$. Thus, we can utilize $\pi\big(a_{ik}^t | s^t\big)\xi_{ij}^t$ to replace $\pi_i\big(a_{ik}^t | s^t\big)$ in $v_i(s^t)$ to relax constraint (5a). For each device, it tries to find the best policy to maximize its cumulative rewards by reaching a Nash equilibrium. It can be reached when no device can obtain a higher reward by changing its policy unilaterally. To satisfy the requirement in Definition 1, we should find the optimal value for state value $v_i(s^t)$. Thus, the following theorem is founded:

Theorem 2 (Nash equilibrium condition). *In our system, in order to reach the global Nash equilibrium among different devices, the state value of each device should meet the following equation:*

$$v_i(s^t) = \prod_{k=1}^{K(i,t)} E_{\xi_{ij}^t \sim \xi, a_{ik}^t \sim \pi_i} q_i(s^t, a_i^t), \forall i \in \mathcal{N}, \tag{7}$$

where the state-action function of device i is:

$$\begin{aligned} q_i(s^t, a_i^t) = E_{\pi_{-i}} \Bigg[& \frac{1}{K(i,t)} \sum_{k=1}^{K(i,t)} r(s^t, a_{ik}^t) \\ & + \gamma \sum_{s^{t+1} \in S} p\big(s^{t+1} | s^t, \boldsymbol{a}^t\big) v_i\big(s^{t+1}\big) \Bigg]. \end{aligned} \tag{8}$$

The proof can be found in Appendix B of Supplementary File, available online.

Although Theorem 2 provides the optimal value of $v_i(s^t)$, we cannot get it beforehand in such a decentralized system. Thus, we need to compute optimal policy $\boldsymbol{\pi}$ by solving the optimization problem. Similar to [29], the process of reaching a Nash equilibrium for our formulated stochastic game can be modeled as an optimization problem related to value function $v_i(s^t)$ and value-action function $q_i(s^t, a_i^t)$. Then, Problem P2 can be transferred into:

$$\begin{aligned} \text{P3:} \quad \min_{\boldsymbol{v}, \boldsymbol{\pi}} g(\boldsymbol{v}, \boldsymbol{\pi}) = \sum_{i=1}^{N} \sum_{t=0}^{\infty} \Bigg[& v_i(s^t) \\ & - \prod_{k=1}^{K(i,t)} E_{\xi_{ij}^t \sim \xi, a_{ik}^t \sim \pi_i} q_i(s^t, a_i^t) \Bigg], \end{aligned} \tag{9}$$

$$\text{s.t.} \quad v_i(s^t) \geq q_i\big(s^t, a_i^t\big), i \in \mathcal{N}. \tag{9a}$$

Constraint (9a) ensures a Nash equilibrium can be achieved among different devices if $v_i(s^t) - q_i(s^t, a_i^t) \geq 0, \forall i \in \mathcal{N}$ holds. Consequently, $g(\boldsymbol{v}, \boldsymbol{\pi}) \geq 0$ holds. According to [30], if $g(\boldsymbol{v}, \boldsymbol{\pi}) = 0$ is founded, a Nash equilibrium can be reached, and $\boldsymbol{\pi}$ is the Nash equilibrium policy.

Next, we prepare to relax Problem P3 by Lagrangian relaxation. Then, we can formulate the Lagrangian dual function of optimization objective (9) in Problem P3 as:

$$L^{t+1} = g(\boldsymbol{v}, \boldsymbol{\pi}) + \sum_{i=1}^{N} \sum_{\tau=0}^{t} \lambda_i^{\tau}(q_i(s^{\tau}, a_i^{\tau}) - v_i(s^{\tau})). \tag{10}$$

Since for any policy $\boldsymbol{\pi}$, equation $g(\boldsymbol{v}, \boldsymbol{\pi}) = 0$ holds, when $v_i(s^t)$ is defined as the form in Theorem 2. Then, Problem P3 can be transferred into:

$$\text{P4:} \quad \max_{\boldsymbol{\lambda}} \min_{\boldsymbol{\pi}} \boldsymbol{L}^{t+1} = \sum_{i=1}^{N} \sum_{\tau=0}^{t} \lambda_i^{\tau}(q_i(s^{\tau}, a_i^{\tau}) - v_i(s^{\tau})). \tag{11}$$

Thus, we merely need to solve Problem P4 to achieve the task scheduling purpose.

Algorithm 1. Pseudo-Code of Expert Policies

```
**Input:** Batch size $B$; intial policies with policy parameter $\theta_0$,
        discriminator parameter $\boldsymbol{\omega}_0$ and value paremeter $\boldsymbol{\phi}_0$.
**Output:** Learned policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}^E$ and expert trajecoties $D$.
for time slot $\tau = 0, 1, 2, ..t$ do
  for device $i = 1, 2, ..N$ do
    for task $j = 1, 2, ..K(i, \tau)$ do
      Get task scheduling results for expert $i$ with size $B$
      based on policy $\pi_i^E$ as $\chi_i^E \sim \pi_{\theta_i}^E$.
    end
  end
end
for device $i = 1, 2, ..N$ do
  Update $\phi_i$ by minimizing the loss:
  $E_{\chi_i^E}\left[\sum_{\tau=0}^{t} E_{K(i,\tau)} r(s^{\tau}, a_{ik}^{\tau}) - v_i^E(s^{\tau}; \phi_i)\right]$.
  Compute policy gradient by equation (16).
  Compute the Fisher metric by equation (17).
  Update $\theta_i$ by using ACKTR role with natural gradient
  $F^{-1}\nabla_{\theta_i} L^E(\theta_i)$.
end
for device $i = 1, 2, ..N$ do
  Get observations $o_i^{\tau}$ from $s^{\tau}$, $\tau = \{0, 1, 2, \ldots, t\}$. for task
  $k = 1, 2, ..K(i, \tau)$ do
    Get expert trajectories by $D_i = \{o_i^{\tau}, a_{ik}^{\tau}\}$, where
    $\tau = \{0, 1, 2, \ldots, t\}$.
  end
end
```

4.4 Expert Policies

To design the multi-agent imitation learning based computation offloading algorithm, we should first design the expert policies that can obtain good performance. We consider that there are N experts interacting with each other in the network, and they can obtain full observations of system states. This is possible because the traditional centralized scheduling policy with good performance can be utilized for demonstration that the learning agent can mimic. Another possible method is to obtain the expert policy offline by applying an algorithm that can achieve optimal performance theoretically. Thus, we merely need to solve Problem P4, and form the demonstrations including state-action pairs for the learning agent to mimic. Then, the learning agent can have some prior experience for task scheduling. For each expert, its optimization problem becomes:

$$\text{P5:} \quad \max_{\lambda_i^{\tau}} \min_{\pi_i} L_i^{t+1} = \sum_{\tau=0}^{t} \lambda_i^{\tau}(q_i(s^{\tau}, a_i^{\tau}) - v_i(s^{\tau})). \tag{12}$$

To solve Problem P5, we can utilize a natural gradient policy, e.g., ACKTR. Traditional policy gradient algorithms generally utilize simple variants of stochastic gradient descent, which update parameters following the direction of policy gradient with inefficient exploration for weight spaces. Different from policy gradient, natural policy gradient follows the steepest descent direction based on Fisher metrics. Thus, ACKTR has a faster convergence speed by applying Kronecker-Factored Approximated Curvature (K-FAC) to approach the natural gradient.

For expert i, two learning networks need to be trained. One is the policy network with parameter θ_i, and the other is the value network with ϕ_i. From the optimization objective (12) in Problem P5, it can be observed that we merely need to minimize $q_i(s^t, a_i^t) - v_i(s^t)$, since λ_i^t can be any constant. For policy π_i^E, the value of $E(q_i(s^t, a_i^t) - v_i(s^t))$ can be minimized. Thus, we define the loss function of the policy model for expert i as:

$$L^E(\theta_i) = E(q_i(s^t, a_i^t) - v_i(s^t)). \tag{13}$$

According to [31], the gradient of $v_i(s^t)$ can be computed by:

$$\nabla_{\theta_i} v_{\pi_i^E}(s^t) = E_{\pi_i}\left[\sum_{a_i^t} q_i(s^t, a_i^t) \nabla_{\theta_i} \log \pi_i(a_i^t | s^t)\right]. \tag{14}$$

Similar to [32], the gradient of $q_i(s^t, a_i^t)$ in our system can be computed by:

$$\nabla_{\theta_i} q_{\pi_i^E}(s^t, a_i^t) = E_{\pi_{-i}}\left[\gamma \sum_{s^{t+1} \in S} p(s^{t+1} | s^t, \boldsymbol{a}^t) \nabla_{\theta_i} v_{\pi_i^E}(s^{t+1})\right]. \tag{15}$$

Then, the policy gradient is defined as:

$$\nabla_{\theta_i} L^E(\theta_i) = E\Big[\nabla_{\theta_i} q_{\pi_i^E}(s^t, a_i^t) - \nabla_{\theta_i} v_{\pi_i^E}(s^t)\Big]. \tag{16}$$

Correspondingly, the Fisher metric can be obtained by:

$$F_i = E_{p_i(B)}\Big[\nabla_{\theta_i} \log \pi_i(a_{ik}^t | s^t)(\nabla_{\theta_i} \log \pi_i(a_{ik}^t | s^t))^{\top}\Big], \tag{17}$$

where $p_i(B)$ is the trajectory distribution, satisfying $p_i(B) = p(s^0) \prod_{t=0}^{T} \prod_{k=1}^{K(i,t)} \pi(a_{ik}^t | s^t) p(s^{t+1} | s^t, \boldsymbol{a}^t)$, where $\sum_{t=0}^{T} K(i, t) = B$. Then, the natural gradient of policy network is $F^{-1}\nabla_{\theta_i} L^E(\theta_i)$.

For the value network with parameter ϕ_i, it can estimate the expected sum of rewards from a given state by following policy π_i^E. The corresponding loss function is defined as:

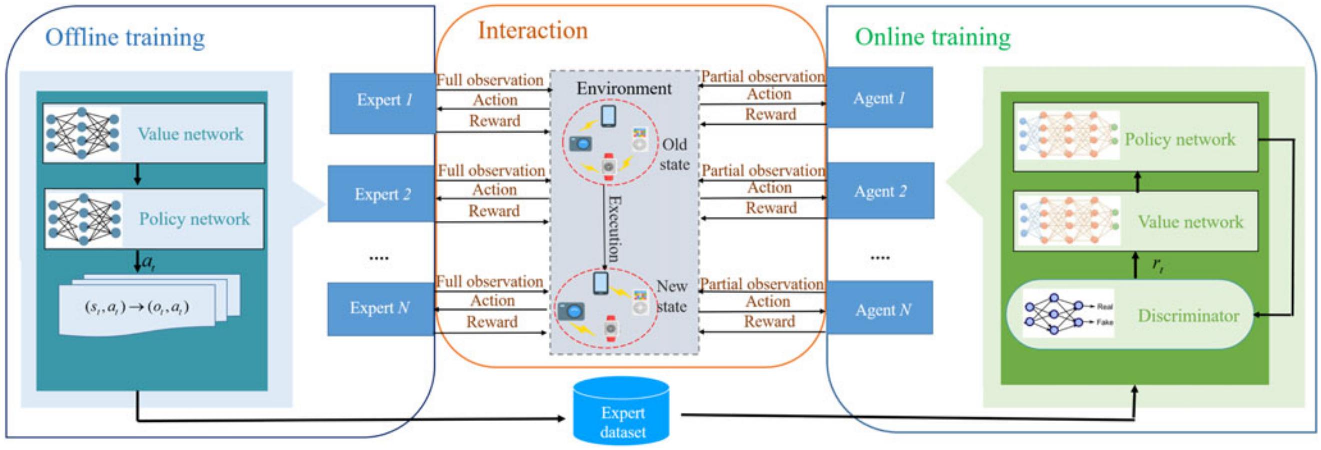


Fig. 3. Structure of the designed algorithm.

$$L^E(\phi_i) = E\left(\sum_{\tau=0}^{t} E_{K(i,\tau)} r(s^\tau, a_{ik}^\tau) - v_i^E(s^\tau;\phi_i)\right). \tag{18}$$

The training process of expert policies in our system can be summarized as follows:

1) *State-action batch collection*: In time slot t, device i utilizes policy π_i^t to generate task scheduling action a_{ik}^t based on system state s^t. Then, the task can be scheduled to the destination according to the selected action, and its reward can be computed based on the corresponding actual transmission and computation delay. For each task, we record its action, reward and the system state in a list. A batch of records with size B can be totally obtained.
2) *Policy and value network training*: After getting state-action batches, the policy and value networks are trained to minimize the loss function in Equations (13) and (18).
3) *Expert demonstration formulation*: Since the devices do not have the full observations of the system, we form expert demonstrations by mapping the instantaneous system state to the partial observation of corresponding devices in the considered pervasive edge computing networks. Then, the observation-action pairs are formed correspondingly to the state-action pairs in each collected batch. The pseudo-code of expert policies is shown in Algorithm 1.

4.5 Agent Policies

In the previous subsection, we introduce the obtained expert policy based on the centralized network environment, i.e., all the experts can obtain the whole system states. However, the expert policies cannot be applied online in the decentralized environment, since devices cannot acquire the whole system states. Thus, we utilize multi-agent imitation learning, where devices (here we call them learning agents) can imitate the expert policies derived in the last subsection and make decisions based on their local observations. The structures of the designed algorithm is shown in Fig. 3.

4.5.1 Imitate from Demonstrations

In imitation learning, two kinds of participants are involved, i.e., experts and learning agents. The experts can provide data set D (also called demonstrations), including I sample trajectories, which form expert policy π_E. The ith trajectory consists of state-action pairs generated by policy π_E, represented by $\langle (s_i^0, a_i^0), (s_i^1, a_i^1), \ldots, (s_i^H, a_i^H) \rangle$. For simplicity and without loss of generality, the length of all trajectories is regarded as the same by H. Based on the provided demonstrations, the agent trains its own policies and then gradually improves them by interacting with surrounding environment. Traditional imitation learning can be regarded as one kind of supervision learning. However, the demonstrations can only be collected for a few samples, and may not contain the states that the agent encounters in the future execution. In this situation, the agents may not make right decisions and cause the degradation of system performance.

GAIL [20] overcomes the drawbacks brought by the traditional supervised imitation learning. Generally, it models the distributions of states and actions included in the expert trajectories, and makes the distributions of state-action pairs visited by the agent close to those of the experts. To model distributions of state-action pairs, GAIL harnesses GAN to train the learning model. It is known that two roles are involved in GAN, i.e., generator G and discriminator D. Generator G generates data, whose distribution is analogous to true data distribution Z, while discriminator D learns to distinguish whether a sample is from true data distribution Z or that generated by generator G. Thus, the objective of GAN is $\min_G \max_D V(G,D) = E_x[\log D(x)] + E_Z[\log(1 - D(G(Z)))]$.

In GAIL, generator G can be regarded as the agent that tries to generate state-action distributions by mimicking those of the expert. Distribution D here is utilized to determine whether actions are generated by the expert or the agent. Followed by the competition between two players, the performance of GAIL can be largely improved. Then, the objective of GAIL is to learn good agent policies through adversarial generated training based on expert demonstrations, i.e.,

$$\begin{aligned}\pi = \arg\min_{\theta}\max_{\omega}\big[&E_{\pi_\theta}\big[\log\big(D_\omega(s^t, \boldsymbol{a}^t)\big)\big]\\ &+E_{\pi_E}\big[\log\big(1 - D_\omega(s^t, \boldsymbol{a}^t)\big)\big]\big] - \eta H(\pi),\end{aligned} \tag{19}$$

where η is a control parameter. Variable $H(\pi)$ is the γ-discounted causal entropy of policy π, where $H(\pi) = E_\pi[-\log \pi(\boldsymbol{a}^t|s^t)]$, and it enhances the exploration operations during the learning process [33].

4.5.2 Decentralized Multi-Agent Imitation Learning

In our work, since there are multiple devices and no centralized management exists, decentralized offloading scheduling strategies should be considered. Thus, each device should learn its own policy independently based on the current obtained information. In addition, a Nash equilibrium should be reached among those devices to maximize their own utilities. To realize multi-agent imitation learning, we first provide the following definition:

Definition 2. *There are N experts in the considered network, followed by N learning agents (devices), each of which mimics the behaviors of its corresponding expert. Leveraging GANs for agent policy acquisition, N generators and N discriminators exist.*

However, learning agent i cannot obtain real-time state s^t, and can only get observation o_i^t. To be specific, the instantaneous transmission and processing queueing status of other indirectly connected devices in time slot t are not observable as described in Section 3.1, and their status of previous time slots is possible to be obtained. Thus, for expert policies, we collect expert dataset $D = \{D_i\}_{i=1}^N$ based on observations o_i^t and a_i^t, $t \in \{0, .., \infty\}$, since they are in the decentralized environment. Similar to [34] where single expert-agent pair is considered, we can define the distribution of observation-action pairs encountered by a learning agent in the multi-agent environment as follows:

Definition 3 (Observation-action pair distribution). *In our considered multi-agent system, the distribution of observation-action pairs for learning agent i to visit can be modeled as* $\rho_{\pi_i,\pi_{-i}} = \prod_{i=1}^N \pi(a_i^t|o_i^t) \sum_{t=0}^{\infty} \gamma^t p(o_i^t = o|\boldsymbol{\pi}^t)$, *where* $p(o_i^t = o|\boldsymbol{\pi}^t)$ *is the distribution possibility of state* s^t *visited by policy* $\boldsymbol{\pi}^t = (\pi_1^t, \pi_2^t, \ldots, \pi_N^t)$.

According to proposition 2in [21], Problem P4 can be casted into the optimization objective of multi-agent imitation learning as:

$$\begin{aligned} \text{P6:} \quad \pi_i = \arg\min_{\theta_i}\max_{\omega_i} \Big[& E_{\pi_{\theta_i}}\left[\log\left(D_{\omega_i}\left(o_i^t, a_i^t\right)\right)\right] \\ & + E_{\pi_i^E}\left[\log\left(1 - D_{\omega_i}\left(o_i^t, a_i^t\right)\right)\right]\Big] - \eta_i H(\pi_{\theta_i}), \end{aligned} \tag{20}$$

where $D_i = \left\{\rho_{\pi_i,\pi_{-i}} : \pi_i \in \Pi\right\}$ can be regarded as the distribution space of all observation-action pairs. From the above objective, we can discover that each learning agent maintains policy network π_i with optimization parameter θ_i, and each discriminator D_i aims to optimize parameter ω_i, while value network v_i is with optimization parameter ϕ_i.

Similar to the expert policy, we utilize ACKTR algorithm to solve Problem P6. The training processing can be summarized as the following steps:

1) *Observation-action batch collection*: We collect trajectory batches of each learning agent based on the constructed training networks instead of training the model directly based on expert trajectories. Obviously, our imitation learning method is model-free, which is different from the traditional ones.
2) *Discriminator training*: Based on the collected expert and agent trajectories of observation-action pairs with length B, we train discriminator D_i, i.e., $\{o_i^t, a_{ik}^t, o_i^{E,t}, a_{ik}^E\}$, by minimizing the loss:

$$\begin{aligned} L(D_i) = & E_{\pi_{\theta_i}}\left[\log\left(D_{\omega_i}\left(o_i^t, a_i^t\right)\right)\right] \\ & + E_{\pi_i^E}\left[\log\left(1 - D_{\omega_i}\left(o_i^t, a_i^t\right)\right)\right], \end{aligned} \tag{21}$$

where $t \in \{0, \ldots, x\}$, $k \in \{1, \ldots, K(i,t)\}$ and $\sum_{t=1}^x K(i,t) = B$ are the training inputs. For prediction, we just need to input observation-action pair $\{o_i^t, a_i^t\}$, and output $\log\left(D_{\omega_i}\left(o_i^t, a_i^t\right)\right)$. The output can be regarded as the predicted reward of the agent action based on expert policies. The predicted reward, instead of the real reward, can be utilized in the agent policy, since it can make the agent policy approach to the expert policy by minimizing the distributions of observation-action trajectories between the expert and the agent.
3) *Value network training*: After getting the predicted reward, we train the value network based on it. The loss function is to minimize the squared difference between the B-step predicted rewards and the state value, i.e.,

$$l(v_i) = E\left(\sum_{\tau=0}^{t}\sum_{k=1}^{K(i,\tau)} \log\left(D_{\omega_i}\left(o_i^\tau, a_{ik}^\tau\right)\right) - v_i(o_i^\tau; \phi_i)\right). \tag{22}$$

4) *Policy network training*: For the policy network, we utilize it to determine the task scheduling result for each task of devices. Generally, the current network status and task specific attributes are inputted, and the policy network outputs the scheduling results of the task. We utilize the policy gradient method to train it, with gradient $\nabla_{\theta_i} J(\theta_i) = \nabla_{\theta_i} v_{\pi_i}\left(o_i^0\right)$.

From the above processes, we can observe that training agent policy merely depends on personal local observations without the need of collecting other agents' updated information. Each agent trains its own model locally with the corresponding expert demonstrations, and it can make decisions independently without considering others' policies. The whole process of agent policy can be found in Algorithm 2.

4.6 Algorithm Analysis

In this subsection, we provide analyses for our designed decentralized computation offloading algorithm. It is proved that the results achieved by the learning agents can reach ϵ-Nash equilibrium. We first present the definition of ϵ-Nash equilibrium as follows:

Definition 4 (ϵ-Nash equilibrium). *In our considered system, there exists variable* $\epsilon > 0$ *to guarantee Markov strategy* $\pi^* = \langle \pi_1^*, \pi_2^*, \ldots, \pi_N^* \rangle$ *satisfies* $v_{\pi_i^*,\pi_{-i}^*}(s^t) \geq v_{\pi_i,\pi_{-i}^*}(s^t) - \epsilon$, *where* $i \in \mathcal{N}$ *and* $s^t \in S$. *Then, the formulated stochastic game is a* ϵ-*Nash equilibrium.*

From the above definition, we can observe that a ϵ-Nash equilibrium is a sub-optimal Nash equilibrium, and each Nash equilibrium can be regarded as a special example of ϵ-Nash equilibrium with $\epsilon = 0$.

Theorem 3 (γ-discounted causal entropy). *In our decentralized computation offloading algorithm, the expression of*

γ-discounted causal entropy of policy π_i is $H(\pi_i) = E_{o_i^t, a_{ik}^t \sim \pi_i}\Big[-\sum_{t=0}^{\infty} \gamma^t \log \prod_{k=1}^{K(i,t)} \pi\big(a_{ik}^t | o_i^t\big)\Big]$.

The proof can be found in Appendix C of Supplementary File, available online.

Theorem 4 (ϵ-Nash equilibrium of our algorithm). *Our designed decentralized computation offloading algorithm can reach a ϵ-Nash equilibrium among different devices, with $\epsilon = \max\{\eta_i |H(\pi_i^E) - H(\pi_i)|\}, i \in \{1, \ldots, N\}$.*

The proof can be found in Appendix D of Supplementary File, available online.

Algorithm 2. Pseudo-Code of the Agent Policy

```
Input: Expert trajoecities $D$; batch size $B$; intial policies with
       policy parameter $\boldsymbol{\theta}_0$, distriminator parameter $\boldsymbol{\omega}_0$ and
       value parameter $\boldsymbol{\phi}_0$.
Output: Learned policy $\boldsymbol{\pi_\theta}$.
for time slot $\tau = 0, 1, , .., t$ do
  for device $i = 1, 2, ..N$ do
    for task $j = 1, 2, ..K(i, \tau)$ do
      Get task scheduling trajecories for device $i$ with size $B$
      based on policy $\pi_i$ as $\chi_i \sim \pi_{\theta_i}$.
      Get observation-action pairs from $D_i$ as $\chi_i^E$.
    end:
  end:
end:
for device $i = 1, 2, ..N$ do
  for task $j = 1, 2, ..K(i, t)$ do
    Solve Problem P5 by the following steps:
    Update $\omega_i$ by minimizing the loss in (21).
    Update $\phi_i$ by minimizing the loss in (22).
    Compute the policy gradient by:
    $\nabla_{\theta_i} L(\theta_i) = E_{\chi_i}\big[q_i(o_i^t, a_i^t)\nabla_{\theta_i}\log \pi_i(a_i^t|o_i^t;\theta)\big] + \eta_i \nabla H(\pi_{\theta_i})$.
    Compute the Fisher metric by:
    $F_i = E_{p_i(B)}\Big[\nabla_{\theta_i}\log \pi_i(a_{ik}^t|o_i^t)(\nabla_{\theta_i}\log \pi_i(a_{ik}^t|o_i^t))^\top\Big]$
    Update $\theta_i$ by using ACKTR role with gradient $F^{-1}\nabla_{\theta_i} L(\theta_i)$.
  end:
end:
```

5 Performance Evaluation

In this section, we present the performance evaluation including simulation setup and results.

5.1 Simulation Setup

Similar to [13] and [35], we set devices randomly on a square area of 1 km^2 with cross and vertical lines. This network topology is with 10^4 cross points formed by the intersecting lines. The considered random network topology can be viewed as a part of the campus map, where mobile terminals held by walkers or riders belong to the same community [36]. Holders move along those lines (can be viewed as roads on campus) based on Manhattan mobility model [37]. Our considered model is time-slot-based due to the high repeatability of individuals' activities. There are 10 to 60 devices moving in the grid topology with speeds between 0 to 20 Km/h [13]. The communication distance of these devices is set between 50m and 100m, and the maximum number of computation tasks generated by each device is 5 per time slot. The task size is between $[100, 10000]$ KB. The required CPU cycle of each task is $[0.2, 1]$ Gigacycles, and the CPU cycle provided by each device is between $[8, 13]$ Gigacycles [38]. The transmission rates of devices can be determined by Shannon formula with bandwidth 10 MHz, and the transmission power is 10 dBm with noise power -172 dBm [39].

We utilize multi-layer perceptions for ACKTR algorithm, with four fully connected layers for policy and value networks, respectively. KFAC Optimizer [40] and Asynchronous Advantage Actor Critic (A3C) technology [41] are utilized to train the learning model. For expert policies, we collect observation-action pairs from 100 to 400 episodes, each of which with 50 tasks to be processed [21]. In addition, four representative algorithms are compared with the designed MILP algorithm:

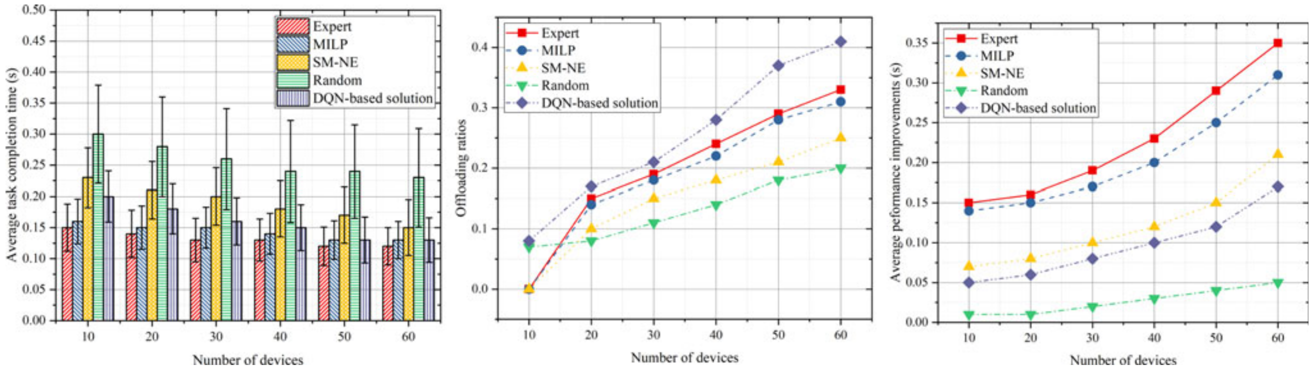
- Expert policies: Experts try to solve Problem P4 by using ACKTR algorithm based on centralized management. The whole instantaneous system states can be observed by each expert.
- SM-NE [13]: It is a decentralized task scheduling algorithm with some required information manged by a centralized server. It is solved by game theory and a Nash equilibrium can be reached. We apply it in our considered pervasive edge computing network.
- DQN-based solution [42]: Similar to our system, it considers the peer-to-peer offloading mode. The main difference is that it includes edge servers, and the device can offload tasks to either neighboring devices or edge servers. In addition, it is also a centralized scheduling algorithm based on Deep Q-Network (DQN).
- Random scheduling: It tries to offload tasks by seeking devices satisfying Problem P1. Once a device is found, the device can be utilized for task processing without further searching.

5.2 Simulation Results

In this subsection, we present the performance results with various metrics.

5.2.1 Impacts of total device number

Fig. 4 illustrates the network performance with different number of devices. The trend of average task completion time is shown in Fig. 4a. We notice that the average task completion time of MILP is much lower than that of SM-NE, DQN-based solution and random scheduling schemes, and a little higher than that of expert policies. For example, when the number of devices is 30, the average task completion time of expert policies, MILP, SM-NE, DQN-based solution and random scheduling schemes is 0.13s, 0.15s, 0.2s, 0.16s and 0.26s, respectively. The reason is that MILP utilizes GAIL to train its policies, and can approach to the performance of experts with an acceptable performance gap. Expert policies try to solve the optimization problem by reaching the Nash equilibrium among different devices based on the whole system states, which is similar to the centralized control method. SM-NE computes the Nash equilibrium based on average task arrival intensities, transmission



(a) Average task completion time (b) Offloading ratios (c) Average peformance improvements

Fig. 4. Performance with different number of devices.

rates, and so on. However, the information cannot be updated timely in pervasive edge computing networks. Thus, the performance of SM-NE cannot exceed that of MILP. Random scheduling algorithm selects devices for offloading randomly, the performance of which cannot be guaranteed. Although DQN-based solution is centralized, it aims to minimize the system cost while ensuring that the task can be executed before its deadline. Thus, the task completion delay cannot be largely minimized since DQN-based solution mainly tries to minimize the system cost.

Fig. 4a also demonstrates the standard deviation of the four algorithms, where expert policies have the minimum deviation, and MILP is the second. This is because MILP comprehensively considers the fairness of devices in the fully decentralized environment by mimicking the expert demonstration, while SM-NE cannot receive the updated information and depends on the outdated information to compute a Nash equilibrium. DQN-based solution focuses on selecting the devices or edge servers that can minimize its system cost instead of minimizing the task completion time. When the number of devices becomes larger, the performance of the five algorithms becomes better. This is because more devices are available in the network, and more chances can be produced for task offloading to minimize their completion time.

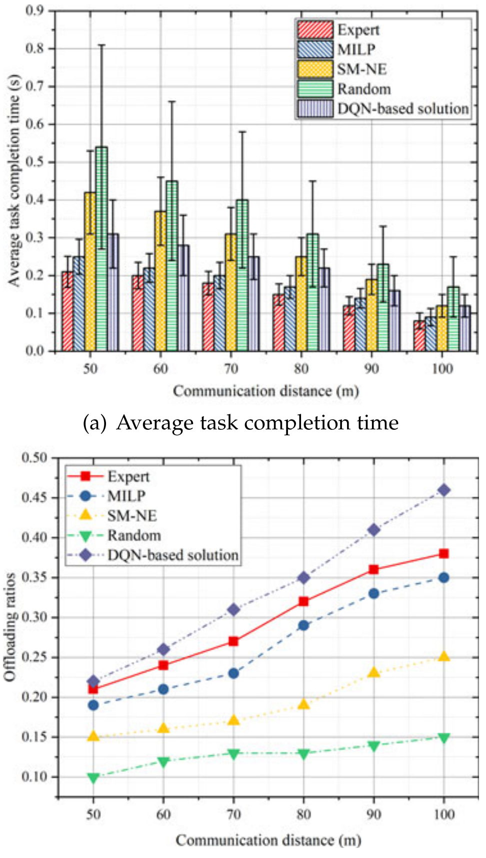
Fig. 4b illustrates the offloading ratios based on different number of devices. Herein, the offloading ratio is defined as the number of offloaded tasks to that of the total generated tasks in the network. It is obvious that the offloading ratio of DQN-based solution is the highest, while those of the expert policy and MILP are higher than those of SM-NE and random scheduling algorithm. For example, when the number of device is 20, the offloading ratios of expert policies, MILP, SM-NE, DQN-based solution and random scheduling schemes are 0.15, 0.14, 0.1, 0.17 and 0.08, respectively. That is to say, MILP can learn a feasible policy to schedule more tasks among different devices based on the expert demonstrations. When the number of devices is lower than 10, the task offloading ratios of expert policies, MILP and SM-NE are lower than those of DQN-based solution and random scheduling algorithm, since there are less chances for tasks to be offloaded to other devices by reaching a Nash equilibrium. DQN-based solution can offload tasks to either the edge server or devices, thus the system cost can be largely decreased even there are less number of devices. The offloading ratios of the five algorithms become large with the increasing number of devices. The reason is that there are more chances for task offloading since more devices are available in the area.

The average performance improvement for the five algorithms is shown in Fig. 4c. It is defined as the performance gap of average task completion time by executing tasks locally and offloading tasks among different devices. It is obvious that the performance of MILP is also the most closest to that of expert policies, and higher than those of the other three algorithms. The reason is that MILP can perform close to the expert policy, since it tries to mimic the behaviors of experts. However, DQN-based solution merely attempts to guarantee that tasks can be executed before their deadline. When the number of devices becomes large, the average performance improvements for the five algorithms also become large. This is because when there are more devices, the average task completion time becomes small as illustrated in Fig. 4a. Thus, the performance gain becomes large.

5.2.2 *Impacts of communication distance*

The performance affected by different communication distances is shown in Fig. 5. The performance trend of the average task completion time along with the change of communication distances is shown in Fig. 5a. When the communication distance becomes larger, the performance of the five algorithm becomes better. For instance, when the communication distance is 60m, the average task completion time is 0.2s, 0.22s, 0.37s, 0.28s and 0.45s, respectively. When the communication distance increases to 80m, the completion time decreases to 0.15s, 0.17s, 0.25s, 0.22s and 0.31s, respectively. This is because when the communication distance between two devices becomes large, they have more chance to communicate with each other, and more available devices can be found for task offloading. Meanwhile, the performance of MILP is the closest to that of the expert policies. The performance standard deviations of the five algorithms also become small, since more available devices for offloading can be found when the communication distance becomes large.

Fig. 5b shows the performance of offloading ratios with different communication distances. The offloading ratio of MILP is higher compared with those of SM-NE and random scheduling algorithms, while lower than that of DQN-based



(a) Average task completion time

(b) Offloading ratios

Fig. 5. Performance with different communication distances.

solution, since MILP can schedule more tasks among different devices to imitate the corresponding expert policies. The offloading ratio rises with the increase of communication distances. When the communication distance is 50m, the offloading ratios of expert policies, MILP, SM-NE, DQN-based solution and random scheduling algorithms are 0.21, 0.19, 0.15, 0.22 and 0.1, respectively. When the communication distance increases to 70m, the offloading ratios of the five algorithms become 0.27, 0.23, 0.17, 0.31 and 0.13, respectively. This is because less opportunity exists for communications among devices when the communication distance is short. Then, less number of tasks can be scheduled and processed among different devices. However, no matter how many devices are in the system, DQN-based solution can allow devices to offload their tasks to the edge server.

5.2.3 Impacts of task size

The average task completion time of the five algorithms based on task sizes is shown in Fig. 6. Herein, the task size refers to the maximum size of the generated tasks in the network. We can notice that when the task size becomes large, the average task completion time of the five algorithms all becomes long. For instance, when the task size is 4000KB, the average task completion time of the four algorithms is 0.11s, 0.13s, 0.16s, 0.15s and 0.17s, respectively. When the task size increases to 6000 KB, the corresponding performance becomes 0.16s, 0.17s, 0.21s, 0.19s and 0.29s, respectively. The reason is that the transmission delay among different devices becomes long when the task size becomes large, which delays the total completion time. Fortunately, the time of our designed MILP algorithm does not increase much due to the designed scheduling mechanism. However, the performance of SM-NE, DQN-based solution and random scheduling algorithms becomes much worse when the task size is large. This is because SM-NE is hard to find available devices to offload tasks in such a fully-decentralized environment, while random scheduling algorithm randomly selects devices to execute tasks, which consumes much time when the task size increases. In addition, DQN-based solution tries to minimize its energy cost first, which ignore minimizing the task completion delay.

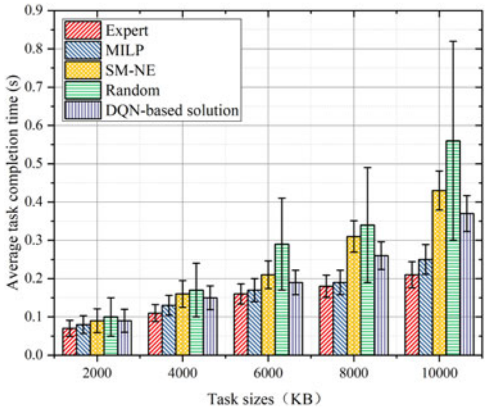


Fig. 6. Average task completion time with different task sizes.

5.2.4 Convergence time

The convergence time of MILP, SM-NE and DQN-based solution is also evaluated based on different number of devices, as shown in Fig. 7. When the number of devices in the network is small, the convergence time of MILP and SM-NE has little difference. When there are more devices in the network, the advantages of the designed MILP algorithm become outstanding. In addition, the convergence time of MILP solution is always the lowest. This is because in MILP, though the full instantaneous network state cannot be observed by each device, the learning agent can imitate the centralized policies by training their own policies in the decentralized environment. The collected expert data can be inputted into GAN and output a predicted reward by minimizing the distribution of observation-action pairs. However, the SM-NE algorithm cannot obtain the immediate information for system states, including the connections of devices and the updated average task arrival speeds, and may make unsuitable decisions based on the outdated information. When the number of devices becomes larger, the time for computing NE becomes longer due to the delay in information synchronization. DQN-based solution is centralized, and it trains the learning model based on the full system states, while outputs the actions for all tasks in each time slot. The state and action spaces are very large, and model training consumes a relatively long time, especially when the number of devices increases.

5.2.5 Impacts of Task Generation Speed

Fig. 8 illustrates the average task completion time of the five algorithms based on the change of task generation speeds.

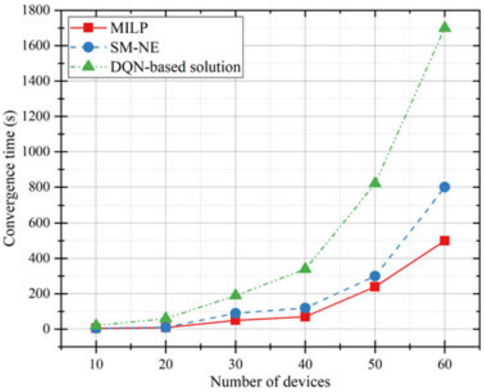


Fig. 7. Convergence time with difference number of devices.

The task generation speed refers to the largest number of tasks generated by each device in each time slot. We can discover that when the task generation speed becomes large, the average task completion time of the five algorithms also increases. This is because there are more tasks in the local queue of each device compared with the situation when the generation speed is low. When the task generation speed is higher than 4 tasks in each time slot, the task completion time increases rapidly due to the additional computation and transmission delay.

5.2.6 Impacts of CPU cycle

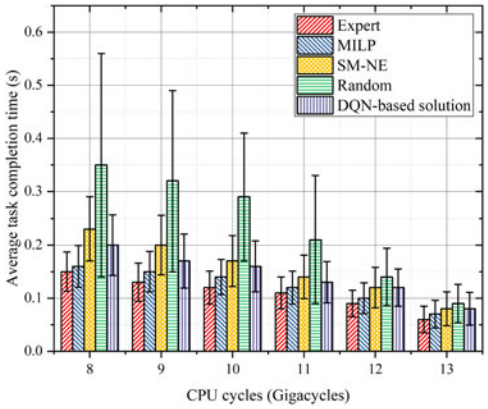


Fig. 9. Average task completion time with different CPU cycles.

Fig. 9 is the performance trend of average task completion time based on different CPU cycles for the five algorithms. The CPU cycle here refers to the maximum CPU cycle one device can provide. The ratios drop when the CPU cycle becomes large. When the CPU cycle is 9 Gigacycles, the average task completion time of expert policies, MILP, SM-NE, DQN-based solution and random scheduling is 0.13s, 0.15s, 0.2s, 0.17s and 0.32s, respectively. When the CPU cycle increases to 12 Gigacycles, the corresponding performance becomes 0.09s, 0.1s, 0.12s, 0.12s and 0.14s, respectively. This is because the computation ability of each device becomes strong when the CPU cycle is large. Thus, the computation delay for processing each task decreases, resulting in less average task completion time. In addition, the performance of our designed MILP algorithm has a little gap with that of the expert policies due to its imitation abilities.

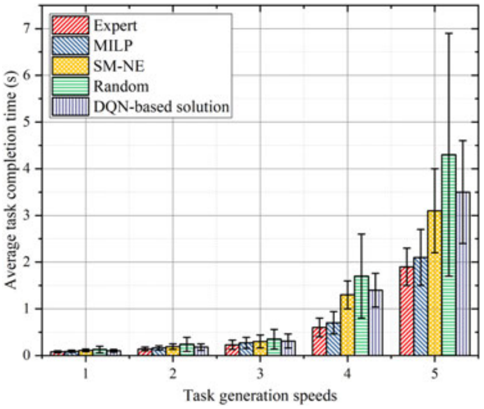


Fig. 8. Average task completion time with different task generation speeds.

6 Conclusion

In this paper, we proposed a multi-agent imitation learning based computation offloading algorithm in pervasive edge computing networks, with the purpose of minimizing the average task completion time for each device. We first established the system model based on pervasive edge computing, and formulated the task scheduling issue based on partial observations as an optimization problem. To the best of our knowledge, this paper is a prior attempt to leverage the generalized adversarial imitation learning in the multi-agent environment for pervasive edge computing. We also proved that our designed algorithm can reach ϵ-Nash equilibrium among different devices. At last, performance results demonstrated the effectiveness of the designed MILP method in average task completion time, convergence time and offloading ratios. In this work, we consider both communication and computing delay to optimize the system performance. In the future, we intend to investigate joint computation, communication and caching scheduling among edge devices to further improve resource utilization.

Acknowledgments

This research was supported in part by the funding from Hong Kong RGC Research Impact Fund (RIF) under Project R5060-19, in part by General Research Fund (GRF) under Project 152221/19E, and in part by the National Natural Science Foundation of China under Grant 61872310, Grant 61971084, and Grant 62001073.

References

[1] Z. Ning *et al.*, "Intelligent edge computing in Internet of vehicles: A joint computation offloading and caching solution," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: 10.1109/TITS.2020.2997832, 2020.

[2] P. Lai *et al.*, "Edge user allocation with dynamic quality of service," in *Proc. Int. Conf. Service-Oriented Comput.*, 2019, pp. 86–101.

[3] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–36, 2019.

[4] D. Zhang *et al.*, "Near-optimal and truthful online auction for computation offloading in green edge-computing systems," *IEEE Trans. Mobile Comput.*, vol. 19, no. 4, pp. 880–893, Apr. 2020.

[5] C. Xu, Y.-F. Zhang, G. Zhu, Y. Rui, H. Lu, and Q. Huang, "Using webcast text for semantic event detection in broadcast sports video," *IEEE Trans. Multimedia*, vol. 10, no. 7, pp. 1342–1355, Nov. 2008.

[6] W. Chen and S. Cai, "Ad hoc peer-to-peer network architecture for vehicle safety communications," *IEEE Communi. Mag.*, vol. 43, no. 4, pp. 100–107, Apr. 2005.

[7] J. Liu *et al.*, "Online multi-workflow scheduling under uncertain task execution time in IaaS clouds," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2906300.

[8] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[9] P. Lai *et al.*, "Quality of experience-aware user allocation in edge computing systems: A potential game," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2020.

[10] S. Sthapit, J. Thompson, N. M. Robertson, and J. R. Hopgood, "Computational load balancing on the edge in absence of cloud and fog," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1499–1512, Jul. 2019.

[11] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments," *IEEE Trans. Mobile Comput.*, vol. 19, no. 4, pp. 852–864, Apr. 2020.

[12] Y. Huang, J. Zhang, J. Duan, B. Xiao, F. Ye, and Y. Yang, "Resource allocation and consensus on edge blockchain in pervasive edge computing environments," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1476–1486.

[13] S. Jošilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.

[14] L. A. Steffenel, M. K. Pinheiro, L. V. Peres, and D. K. Pinheiro, "Strategies to implement edge computing in a P2P pervasive grid," in *Fog Computing: Breakthroughs in Research and Practice*, Hershey, PA, USA: IGI Global, 2018, pp. 142–157.

[15] C. Cicconetti, M. Conti, and A. Passarella, "Low-latency distributed computation offloading for pervasive environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–10.

[16] Q. Meng, K. Wang, B. Liu, T. Miyazaki, and X. He, "QoE-based big data analysis with deep learning in pervasive edge environment," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.

[17] X. Song, Y. Huang, Q. Zhou, F. Ye, Y. Yang, and X. Li, "Content centric peer data sharing in pervasive edge computing environments," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 287–297.

[18] X. Wang, Z. Ning, S. Guo, and L. Wang, "Imitation learning enabled task scheduling for online vehicular edge computing," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2020.3012509.

[19] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 627–635.

[20] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 4565–4573.

[21] J. Song, H. Ren, D. Sadigh, and S. Ermon, "Multi-agent generative adversarial imitation learning," in *Proc. Advances Neural Inf. Process. Syst.*, 2018, pp. 7461–7472.

[22] M. Liu *et al.*, "Multi-agent interactions modeling with correlated policies," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–20.

[23] X. Wang, Z. Ning, and L. Wang, "Offloading in Internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Inform.*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018.

[24] Thomas and U. Marlin, "Queueing systems," *SIAM Rev.*, vol. 18, no. 3, pp. 512–514, 1977.

[25] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 5279–5288.

[26] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Advances Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[27] H. Li, L. Lai, and H. V. Poor, "Multicast routing for decentralized control of cyber physical systems with an application in smart grid," *IEEE J. Sel. Areas Commun.*, vol. 30, no. 6, pp. 1097–1107, Jul. 2012.

[28] R. Madan and S. Lall, "Distributed algorithms for maximum lifetime routing in wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 5, no. 8, pp. 2185–2193, Aug. 2006.

[29] J. Filar and K. Vrieze, *Competitive Markov Decision Processes*. Springer: Springer, 2012.

[30] H. Prasad and S. Bhatnagar, "A study of gradient descent schemes for general-sum stochastic games," 2015, *arXiv:1507.00093*.

[31] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 5279–5288.

[32] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Advances Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[33] M. Bloem and N. Bambos, "Infinite time horizon maximum causal entropy inverse reinforcement learning," in *Proc. IEEE Conf. Decis. Control*, 2014, pp. 4911–4916.

[34] U. Syed, M. Bowling, and R. E. Schapire, "Apprenticeship learning using linear programming," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 1032–1039.

[35] P. Zhou, C. Wang, and Y. Yang, "Self-sustainable sensor networks with multi-source energy harvesting and wireless charging," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1828–1836.

[36] Z. Ning, F. Xia, X. Hu, Z. Chen, and M. S. Obaidat, "Social-oriented adaptive transmission in opportunistic internet of smartphones," *IEEE Trans. Ind. Inform.*, vol. 13, no. 2, pp. 810–820, Apr. 2017.

[37] G. Walunjkar and A. K. Rao, "Simulation and evaluation of different mobility models in disaster scenarios," in *Proc. IEEE Int. Conf. Recent Trends Electron. Inf. Commun. Technol.*, 2019, pp. 464–469.

[38] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2017.

[39] L. Chen and J. Xu, "Task replication for vehicular cloud: Contextual combinatorial bandit with delayed feedback," in *Proc. Int. Conf. Comput. Commun.*, 2019, pp. 748–756.

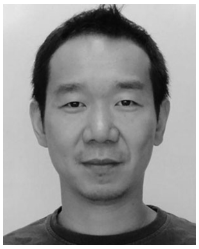
[40] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2408–2417.

[41] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[42] Q. Luo, C. Li, H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2020.2983660.

Xiaojie Wang (Member, IEEE) received the MS degree from Northeastern University, China, in 2011, and the PhD degree from the Dalian University of Technology, Dalian, China, in 2019. From 2011 to 2015, she was a software engineer with NeuSoft Corporation, China. Currently, she is a postdoctor with the Hong Kong Polytechnic University. Her research interests include wireless networks, mobile edge computing, and machine learning.

Zhaolong Ning (Senior Member, IEEE) received the PhD degree from Northeastern University, China, in 2014. He was a research fellow with Kyushu University from 2013 to 2014, Japan. Currently, he is an associate professor with the Dalian University of Technology and a distinguished professor with the Chongqing University of Posts and Telecommunications. His research interests include Internet of Things, mobile edge computing, and resource management. He has published more than 120 scientific papers in international journals and conferences. He has served as an associate editor or guest editor of several journals, such as the *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Social Computational Systems*, *Computer Journal*, IJCS and so on. He is the outstanding associate editor of *IEEE Access* 2018. He is elected to be the Young Elite Scientists Sponsorship Program by CAST and Hong Kong Scholar.



Song Guo (Fellow, IEEE) received the PhD degree in computer science from the University of Ottawa and was a professor with the University of Aizu. He is a full professor with the Department of Computing, The Hong Kong Polytechnic University. His research interests include big data, cloud computing and networking, and distributed systems with more than 400 papers published in major conferences and journals. His work was recognized by the 2016 Annual Best of Computing: Notable Books and Articles in Computing in ACM Computing Reviews. He is the recipient of the 2017 IEEE Systems Journal Annual Best Paper Award and other five best paper awards from IEEE/ACM conferences. He was an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and an IEEE ComSoc distinguished lecturer. He is currently on the editorial board of the *IEEE Transactions on Emerging Topics in Computing*, *IEEE Transactions on Sustainable Computing*, *IEEE Transactions on Green Communications and Networking*, and *IEEE Communications*. He also served as general, TPC and symposium chair for numerous IEEE conferences. He currently serves as an officer for several IEEE ComSoc Technical Committees and a director in the ComSoc Board of Governors.