Resource Allocation in IoT Edge Computing via Concurrent Federated Reinforcement Learning

Zhu Tianqing, Wei Zhou, Dayong Ye, Zishuo Cheng, and Jin Li, *Senior Member, IEEE*

***Abstract*—Resource allocation is a fundamental research issue in IoT edge computing, and reinforcement learning is fast becoming a common solution. The majority of the current techniques involve decision makers who determine how and where resources should be distributed. In a standard cloud system, this decision maker is a central server. In an edge system, the decision maker is an edge host. Both approaches have drawbacks. Edge hosts do not always have access to enough global information to create the most optimal resource allocation strategy. Central servers do but at the cost of privacy. A solution is needed that can do both. This article, therefore, presents a novel resource allocation method called concurrent federated reinforcement learning. The scheme inherits the privacy protection of federated learning, the complex problem solving power of reinforcement learning and adds concurrency in the form of joint decision making so the resource allocation strategies work to the benefit of the global system. The experiments demonstrate that the approach provides the state-of-the-art performance in system-wide utility, speed of task completion, and resource utilization.**

***Index Terms*—Deep reinforcement learning, privacy preservation, resources allocation.**

I. Introduction

EDGE computing was originally conceived as a solution to the cyber sprawl of IoT systems [1]. As IoT systems began to support more and more devices, central servers could no longer cope with the computational burden in a timely fashion. Edge computing pushed some of the processing onto the end devices to reduce dependency on the center of the cloud. Although an ingenious solution, it did not come without raising new impediments to the future development of cloud systems. One of the most important of these is how to optimally allocate computing resources between the edge nodes and the central cloud [2].

Manuscript received March 20, 2021; revised May 28, 2021; accepted June 1, 2021. Date of publication June 14, 2021; date of current version January 7, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61972366; in part by the National Key Project of China under Grant 2020YFB1005700; and in part by the National Natural Science Foundation of China for Joint Fund Project under Grant U1936218. *(Corresponding author: Jin Li.)*

Zhu Tianqing and Wei Zhou are with the School of Computer Science, China University of Geosciences, Wuhan 430074, China (e-mail: tianqing.zhu@ieee.org).

Dayong Ye and Zishuo Cheng are with the Centre for Cyber Security and Privacy and the School of Computer Science, University of Technology Sydney, Sydney, NSW 2007, Australia (e-mail: dayong.ye@uts.edu.au; zishuo.cheng@uts.edu.au).

Jin Li is with the School of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou 510006, China, and also with Pazhou Lab, Guangzhou 510330, China (e-mail: jinli71@gmail.com).

Digital Object Identifier 10.1109/JIOT.2021.3086910

The two main approaches to this problem both involve decision makers. The first and more traditional is an omniscient controller who makes decisions based on global information [3], whether that be a central server, an edge machine, or third-party system. The second and more recent approach is to leave the decisions to the edge hosts or even the end device, and let each determine the best resource allocation for their own needs based on local information [4]. Global methods achieve good overall throughput, which works well for system administrators wishing to maximize use of their infrastructure. However, they leave the system vulnerable to breaches of privacy and security. Local methods relieve the burden on the central server and typically provide good response times. But system-wide efficiency tends to be poor. The infrastructure tends not to be used to its maximum capacity and bottlenecks can still occur.

Our goal is to develop a resource allocation method that provides global optimality without sacrificing the privacy of any party in the system. This is a challenging objective because, to preserve privacy, a party has to either hide or obfuscate their private information [5] and, without full and clear information, global optimality can be difficult to achieve. Our answer is to develop a resource allocation scheme that adds joint decision making to the amalgamation of federated reinforcement learning. We call the scheme concurrent federated reinforcement learning, or CFRL-based for short.

Federated learning was first proposed for mobile computing environments. It gave mobile devices a way to collaboratively learn a prediction model without violating the privacy of end users caused by sharing training data [6]–[8]. In contrast, reinforcement learning was conceived to solve highly complex sequential decision-making problems. Reinforcement learning models have since been integrated with deep neural networks to create deep reinforcement learning, which has proven to meet some of the toughest challenges in strategic AI, such as mastering the game of Go [9]. Amalgamating the two schemes into federated reinforcement learning has combined the advantages of both paradigms, giving decision makers enormous power to reach strategically optimal decisions without compromising the privacy of each party's training data [10]. However, existing federated reinforcement learning methods share a common drawback in that each client's local model must be uploaded to the server to form a global model [11]–[15]. A white hat study by Geiping *et al.* [16] shows that an adversary can reconstruct the input data of a model using only the model's gradient. Therefore, since the server holds each client's local model, anyone with access to the server could use the Geiping method to reconstruct the training data of any and all clients contributing to the system.

To overcome this critical drawback, we devised concurrency as a new category of decision making, where the server and the edge hosts make decisions about resource allocation together. Furthermore, in the CFRL-based algorithm, the edge hosts only share the output of their local model with the server so neither the server nor the edge hosts have any knowledge of the other hosts' input data or local model parameters. The server guides learning by dividing its rewards between the edge hosts, so, again, neither the server nor the edge hosts have any knowledge of the other hosts' private decision-making policies. While data privacy can never be 100% guaranteed, this scheme offers stronger privacy than standard federated learning, and it is not vulnerable to model inference attacks.

In summary, our work makes the following main contributions to the literature.

1) This is the first resource allocation method to consider the privacy of both the edge hosts and the server.
2) We introduce the notion of concurrency to federated reinforcement learning, where the agents and the server both make decisions jointly but neither party ever shares their models, only the outputs and rewards. This novel algorithm can be applied to other similar collaborative scenarios.
3) The experiments that show the method delivers outstanding performance even with only limited information shared by each party.

The remainder of this article is organized as follows. The next section reviews and compares our research with similar work. Section III gives the preliminaries for the reinforcement and federated learning schemes, followed by the details of the resource allocation method in Section IV. Theoretical analyses of privacy and utility are given in Section V. The experiments and comparisons to the current state-of-the-art approaches are presented in Section VI, and the article concludes in Section VII with a brief summary of the article and pointers to our future work.

II. Related Work

Reinforcement learning, as a tool for allocating resources in edge computing environments, has been investigated extensively. Therefore, we have only included studies in this review that are closely related to our research. A thorough review of resource management for edge computing can be found in [2] and [17].

As mentioned, reinforcement learning methods for resource allocation mainly fall into two categories: 1) those with local decision makers and 2) those with global decision makers. Relevant studies in each category are discussed next.

A. Local Decision Makers

In methods with local decision makers, resource allocation strategies are typically formulated by each end user or an edge host using only local information. Each entity, therefore, has an individual aim to maximize its own utility, but such that the aggregate utility of all entities can also be maximized—where possible. Where there are conflicts of utility, independently maximizing one's own generally takes precedence that aggregate utility suffers.

Chen *et al.* [18] considered a mobile user offloading tasks to base stations. They modeled the problem of selecting base stations as a Markov decision process (MDP) and solved it via deep reinforcement learning. The learning algorithm's inputs comprise the mobile user's task queue, base stations' energy queues, and the channel quality between the user and the base stations. The outputs policy guides phones in which base stations to select.

Liu *et al.* [19] proposed a deep reinforcement learning algorithm for resource allocation with vehicles as the edge device. Within the network, each vehicle acts as a mobile edge server, offering computation services to nearby users. When a user requires a computation task, nearby vehicles make a decision about whether the task should be executed by the user or the vehicle server, or whether it should be sent to a fixed edge server to maximize the network utility.

Liu *et al.* [20]–[22] researched resource allocation using agent-based reinforcement learning. In [20], they use a reinforcement learning approach to model each end device as an agent that can make decisions on whether to offload computation tasks to an edge host(s). In [21], they first group end users into clusters based on users' priorities. Then, in each cluster, each user, modeled as an agent, applies deep reinforcement learning to make a series of decisions on task offloading. In [22], IoT users (i.e., agents) make decisions on computation offloading in an aggregated manner, where each agent learns independently with an overall computation capacity signal shared by the central gateway.

Xiong *et al.* [4] proposed a deep reinforcement learning approach for resource allocation in IoT edge computing to minimize the average completion time of tasks and the average number of requested resources. Each edge host independently learns how to allocate computing resources to each of its tasks. The computing resources are divided into sliding windows by time. To improve learning performance, they introduced multiple experience replay memories into the classical deep Q-network (DQN) algorithms, where each memory corresponds to a resource request.

B. Global Decision Makers

In global information methods, decisions are usually made by a cloud server or an omniscient controller [23]–[29], where the aim is to maximize the overall utility of the system. However, as a controller must collect status information from all entities to create an optimal resource allocation policy, privacy is an issue.

Wei *et al.* [3] jointly optimized content caching, computation offloading, and radio resource allocation in edge computing, solving their joint decision-making problem by minimizing the average end-to-end delay using actor–critic deep reinforcement learning. The critic adopts a deep neural network to estimate the value functions of states and actions, while the actor employs another deep neural network to represent a parameterized stochastic policy so as to improve the final policy. With this approach, falling into a local optimum is seldom a problem.

Qiu *et al.* [30] integrated blockchain techniques into deep reinforcement learning for resource allocation to guarantee that IoT device data could not be tampered with. In addition to the status of each IoT device, the reward and cost of the blockchain mining are also taken into account during learning. The decision, output by the learning algorithm, not only includes how to offload tasks but also whether to conduct blockchain mining.

Alqerm and Pan [31] proposed an enhanced online Q-learning scheme for resource allocation to achieve both maximum utility of the IoT system and fairness among IoT applications. To achieve the two goals, they quantified fairness using the envy-free feature [32] and integrated fairness into the objective function. Then, both utility and fairness can be optimized simultaneously by using the Q-learning scheme to optimize the objective function.

Both [13] and [33] attempted to apply federated learning to improve reinforcement learning performance with individual IoT devices. In their studies, federated learning and reinforcement learning are conducted separately. The edge hosts use reinforcement learning to learn a local model, which is then uploaded to the server. The server aggregates these local models into a global model and shares the global model back to the edge hosts. CRFL, by comparison, allows edge hosts and the cloud server to learn concurrently by sharing only the outputs of models and the rewards received. In this way, the model privacy of both edge hosts and the cloud server can be preserved.

C. Discussion of Related Work

Local methods tend not to provide system-wide speed or efficiency because the entities making the decisions do not have global information. Conversely, global methods can manage resources in an optimal way to ensure all tasks across an entire system are completed in as timely a fashion as possible. However, privacy is sacrificed in the process. Additionally, some of the federated learning approaches prioritize learning performance over privacy, negating much the original benefit of using the paradigm. Some research proposed to directly use federated learning as a privacy-preserving method for service placement and resource allocation in edge computing [34]. However, as we will discuss in Section V-B, edge hosts using federated learning are still vulnerable to adversarial activities. By combined federated learning with reinforcement learning and, additionally, integrating concurrent decision making into the schema, our CFRL-based algorithm provides a means to global optimality while providing a strong privacy guarantee than traditional federated learning.

III. Preliminaries

A. Reinforcement Learning

Reinforcement learning is based on a trial-and-error-style learning paradigm that was primarily designed to solve sequential decision-making problems. Formally, the sequential decision-making process can be modeled as an MDP [35], expressed as a tuple $\langle S, A, T, R\rangle$, where S is the set of states, A is a set of actions available to the agent, T is the transition function, and R is the reward function.

At each step, an agent observes the state $s \in S$ of the environment, and selects an action $a \in A$ based on its policy π. After performing the action, the agent receives a real-valued reward r, and the environment changes to a new state $s' \in S$. The agent then updates its policy π based on the reward r and the new state s'. The policy π can be represented as a Q-function $Q(s, a)$, which estimates the reward that the agent will earn in state s by performing action a. Via this strategy, the agent gradually accumulates knowledge and improves its policy to maximize its total long-term expected reward.

The Q-function $Q_\pi(s, a)$ for a particular policy π is defined as

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[\sum_{t=1}^{\infty} \gamma \cdot r(s_t, a_t) | s_t = s, a_t = a\right] \quad (1)$$

where γ is the discount factor, and the optimal policy π^* is the policy that yields the largest value of the Q-function, i.e.,

$$Q^*(s, a) = \max_\pi Q_\pi(s, a). \quad (2)$$

Thus, finding the optimal policy π^* is equivalent to deriving the optimal Q-function $Q^*(s, a)$, which can be done using the Bellman equation

$$Q^*(s, a) = \mathbb{E}_\pi\left[r(s_t, a_t) + \gamma \max_{a'} Q^*(s', a') | s_t = s, a_t = a\right]. \quad (3)$$

Accordingly, the value of the Q-function can be computed via iterative updates as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]. \quad (4)$$

Note that nondeep versions of reinforcement learning cannot handle highly complex problems, such as those involving a large number of states and/or actions. This is because the Q-table, which stores the Q-values of the state–action pairs $Q(s, a)$, becomes too large to update and maintain in a timely manner. Mnih *et al.* [36] thought to incorporate a DQN into the scheme, giving rise to deep reinforcement learning. In deep reinforcement learning, the Q-table is approximated with a neural network, i.e., $Q(s, a; \theta)$, which is far less expensive than maintaining a large Q-table. Here, θ denotes the weights of the network, the state s is the input, and the output is a vector of Q-values, with each value corresponding to an action a. This new, more efficient architecture is capable of handling highly complex problems.

To learn the optimal value of $Q(s, a; \theta)$, a mean-squared error loss function $L(\theta)$ is used to update weights θ

$$L(\theta) = \frac{1}{|B|}\sum_{e \in B}\left[\left(r(s, a) + \gamma \max_{a'} Q\left(s', a'; \theta\right) - Q(s, a; \theta)\right)^2\right] \quad (5)$$

where $e = \langle s, a, r(s, a), s'\rangle$ is an experience sample that shows a state transition with a reward; and B is a batch of experience

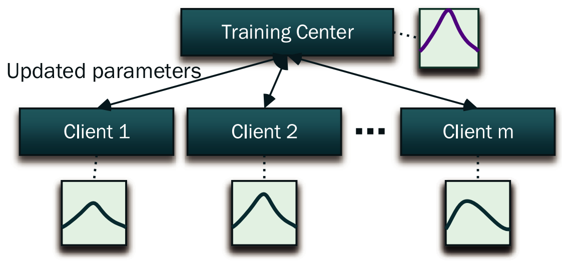

Fig. 1. Typical federated learning framework. In this framework, each client trains a local model with local data. The clients' local data can have different data distributions. Then, the updated parameters of each local model are sent to the server, also known as a training center. The center collects these updates to form a global model and distributes the global model to the clients to help them update their local models.

samples used to train the DQN. Minimizing $L(\theta)$ is identical to reducing the mean-squared error in a Bellman equation.

B. Federated Learning

Federated learning was first proposed by Google in 2017 as an alternative approach to standard centralized machine learning [37]. In federated learning, individual users collaboratively learn a shared prediction model without sharing the private data on each user's local device as training data. While not foolproof, this paradigm does guard against several different types of privacy attacks and so has been adopted in several real-world applications [7].

Fig. 1 shows the structure of a typical federated learning framework. The training center distributes a general learning model to all the devices in the network that has been trained on general data to complete a broad task—say, image processing. In the first learning iteration, each client, also known as user, downloads the general model from the training center to their local device. They improve it by learning from their own local data and send a summary of the improvements back to the server as a small update via a secure communications channel. The cloud server compiles all the updates into a shared model that is improved wholesale. This process repeats until the clients download each updated global model in subsequent steps. During this entire process, no client data are ever shared with either the training center or another client; it stays on the local device.

However, not sharing data is not enough to protect the privacy of data in federated learning, as advanced attack methods can utilize model parameters to reconstruct training data [16]. Existing defense methods include differential privacy [38], secure aggregation [39], and secure multiparty computation [40]. However, differential privacy may significantly reduce the accuracy of resulting models [41]. Secure aggregation may require huge computation cost [16]. Secure multiparty computation may incur extra communication overhead caused by passing tokens among clients [40]. These drawbacks of existing defense methods motivate us to develop a secure and efficient federated learning framework for resource allocation in IoT edge computing.

TABLE I
MEANING OF EACH NOTATION

Notations	Meaning
S	state space
A	action/strategy space
T	transition function
R	reward function
n_e	the number of edge hosts
$\mathbf{C}$	resource allocation status matrix of an edge host
$\boldsymbol{\tau}$	resource requirement vector of an edge host
m	the number of resources possessed by an edge host
n	the length of time sliding window
c^{τ_i}	the number of resources allocated to task τ_i
t	the index of time step
t^{τ_i}	the starting time step of task τ_i
n_τ	the number of resources possessed by the server
$d_1, \ldots, d_{n_e}$	a division of the server's resources reserved for n_e edge hosts
u^{τ_i}	the utility of completing task τ_i
$t_q^{\tau_i}$	the number of time steps that task τ_i stays in the queue
t_0	the number of time steps of sending a task to the server
r_e	reward of an edge host
r_s	reward of the server
λ	the mean rate of a Poisson distribution

C. Federated Reinforcement Learning

Combining deep reinforcement learning with federated learning yields federated deep reinforcement learning [10] as a new computing paradigm, where each user trains a local DQN and the center trains a general DQN. In each learning iteration, the center sends a policy to each user. By following this policy, each user performs an action and receives a reward. Each user then updates her local DQN using the received reward. Also, each user sends her reward signal back to the center, which updates the general DQN based on these reward signals.

IV. CONCURRENT FEDERATED REINFORCEMENT LEARNING METHOD

A. Overview of Our Method

Consider an edge computing system consisting of one server and multiple edge hosts. Each edge host services a number of IoT devices that regularly upload tasks to be processed. The tasks are stored in a queue on the edge host as they wait to be scheduled. The edge host monitors and manages its resource allocations in collaboration with the server based on the tasks in the queue, scheduling tasks according to the resources it has or offloading work to the server. The objective is to complete all the tasks in the entire system as soon as possible.

This global objective is formulated as an MDP process with three variables: 1) states; 2) actions; and 3) rewards. For the edge hosts, the states are their status, and the actions are their resource allocation strategies. Given speedy completion is the objective, rewards are given on the basis of average task completion time. For the server, the states consist of the resource allocation strategies formulated by the edge hosts, and the actions are the server's resource division strategies. The server's reward is based on the number of resources reserved for the edge hosts and the number of tasks completed on behalf of the edge hosts.

Over the next sections, we describe the process flow, each of the variables and the algorithms that bring the framework together. An overview of the scheme is illustrated in Fig. 2, and the notations used throughout are summarized in Table I.

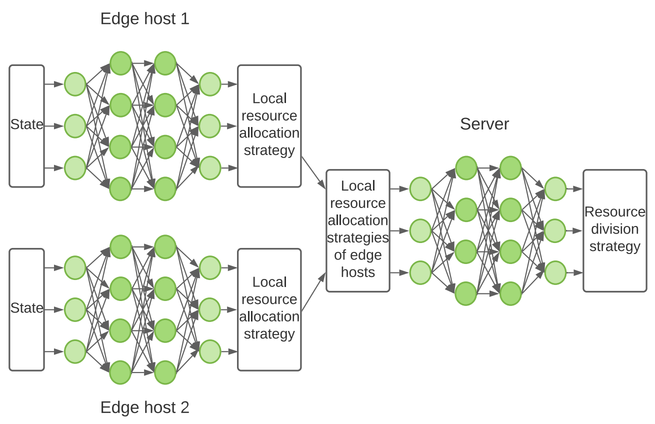
Edge host 1
State
Local resource allocation strategy
Server
Local resource allocation strategies of edge hosts
Resource division strategy
State
Local resource allocation strategy
Edge host 2

Fig. 2. CFRL-based resource allocation framework. Each edge host develops a local resource allocation strategy and shares it with the server. The server formulates a resource division strategy and reserve resources for each edge host accordingly.

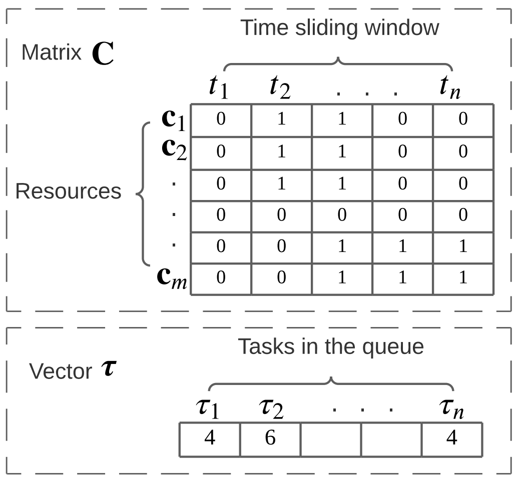

Fig. 3. State(s) of an edge host.

B. Learning Process

Each learning round has three phases: 1) the strategy creation phase; 2) the strategy execution phase; and 3) the model update phase. In the strategy creation phase, each edge host formulates a local resource allocation strategy and shares it with the server. The server then generates a resource division strategy to reserve resources for each edge host. Both types of strategies are built using a deep reinforcement learning algorithm. In the strategy execution phase, the edge hosts either allocate their own resources to the tasks in their queue following the created strategy, or they can send a task to the server to perform using its reserved resources. If a task was successfully allocated, the edge host receives a reward. Likewise, the server also receives a reward based on how many resources it reserved and how many tasks it completed. In the final model update phase, both the edge hosts and the server use their rewards received to update their local DQN.

C. State

The status of each edge host is described as a state s, which is a two-tuple formally defined as

$$S = \{s | s = \langle \mathbf{C}, \boldsymbol{\tau} \rangle\} \tag{6}$$

where $\mathbf{C}$ is a matrix denoting the resource allocation status, and $\boldsymbol{\tau}$ is a vector designating the resource requirement of each task. Thus, a state s can be expressed in matrix form as per Fig. 3.

Matrix $\mathbf{C}$ is an $m \times n$ matrix, where m is the number of resources possessed by the edge host and n is the length of sliding window. Each value in matrix $\mathbf{C}$ is a binary flag, where 0 means the resource is available in the given time step and 1 means it has been allocated. Each row in $\mathbf{C}$ denotes the scheduling of a resource $\mathbf{c}_i$, $1 \leq i \leq m$, over the future n time steps. For example, in the first row of $\mathbf{C}$, resource $\mathbf{c}_1$ has been allocated and will be used in time steps t_2 and t_3. Note that these resources, $\mathbf{c}_1, \ldots, \mathbf{c}_m$, have the same functionality. We have just used different subscripts for the simplicity of presentation. The columns of matrix $\mathbf{C}$ represent time steps in the sliding window, which move left by one column as time passes. The length of a time step, e.g., a second, a millisecond, and a day, depends on specific application.

The vector $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_n]$ is an n-dimensional vector representing the resources needed to complete n tasks in the queue. Each score in the vector indicates the number of resources required by the corresponding task. For example, in vector $\boldsymbol{\tau}$ in Fig. 3, the number of resources required by the first task τ_1 is 4: $\tau_1 = 4$, while the number of resources required by the second task τ_2 is 6: $\tau_2 = 6$. For ease of presentation, we have used τ_i to represent both the name of the ith task and the number of resources required by the ith task.

D. Action

As shown in, the edge hosts have a different action space to the server. Each is described in the following using a very simple example where resources only need to be allocated to one task per time step. In reality, a resource allocation action is a strategy that could involve many of the tasks in the edge host's queue.

1) Actions of Each Edge Host: Formally, an edge host action e is a two-tuple, defined as

$$A_e = \{a | a = \langle c, t \rangle\} \tag{7}$$

where $c \in \{1, \ldots, m\}$ and $t \in \{1, \ldots, n\}$. Thus, an action $a = \langle c, t \rangle$ represents a series of resource allocations c available positions in matrix $\mathbf{C}$ from the tth time step until the time step in which the task is completed. For example, assume that the task τ_1 needs six resources, i.e., $\tau_1 = 6$. One possible action would be $a = \langle 3, 1 \rangle$, which means that the task is given three resources in time step 1 and a further three resources in time step 2, as shown in Fig. 4.

For a given task τ_i, $1 \leq i \leq n$, the action set is $A_e(\tau_i) = \{a^{\tau_i} | a^{\tau_i} = \langle c^{\tau_i}, t^{\tau_i} \rangle \cup \langle 0, 0 \rangle\}$, where $c^{\tau_i} \in \{c | \tau_i \ mod \ c = 0\}$, and $t^{\tau_i} = \{t | \mathbf{C}_t = \mathbf{0}\}$. Here, $\mathbf{C}_t$ is a submatrix of $\mathbf{C}$. The number of rows in $\mathbf{C}_t$ is c^{τ_i}, while the number of columns is (τ_i / c^{τ_i}). The first entry of $\mathbf{C}_t$ starts at the lowest available index, as shown in Fig. 4. The action $\langle 0, 0 \rangle$ means that the task is sent

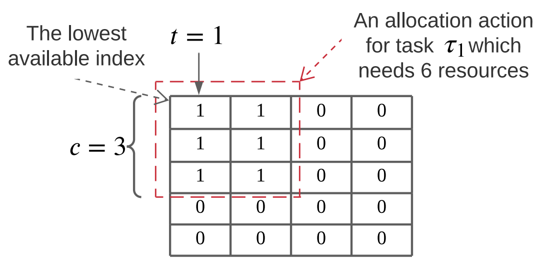
The lowest available index
t = 1
An allocation action for task τ1 which needs 6 resources
c = 3
1 1 0 0
1 1 0 0
1 1 0 0
0 0 0 0
0 0 0 0

Fig. 4. Example edge host action for task τ_1. This action requires six resources.

to the server, as the edge host does not have enough available resources for the task.

2) Actions of Server: The server's aim is to complete the tasks that cannot be handled by the edge hosts. Thus, actions by the server are, in effect, resource divisions. For example, assume that the server has ten resources and there are two edge hosts. A possible action by the server would be $\langle 3, 7\rangle$, which means that three resources are allocated to the first edge host and seven to the second. Note that the server does not have to allocate all its resources. Hence, $\langle 2, 5\rangle$ is another possible action.

Given that there are n_e number of edge hosts and the server has n_r resources, the action set of the server is $A_s = \{a_s | a_s = \langle d_1, \ldots, d_{n_e}\rangle\}$, where each $0 \leq d_i \leq n_r$, $i \in [1, n_e]$, and $\sum_{i=1}^{n_e} d_i \leq n_r$.

E. Reward

As mentioned, the overall objective is to complete all tasks system wide as soon as possible. Thus, rewards for the edge hosts are given based on the number of time steps used to complete a task. Formally, given a state s and an action $a^{\tau_i} = \langle c^{\tau_i}, t^{\tau_i}\rangle$, where $i \in [1, n]$, the reward of an edge host r_e is defined as

$$r_e = \begin{cases} u^{\tau_i} - \left(t^{\tau_i} + \frac{\tau_i}{c^{\tau_i}}\right), & \text{if task } \tau_i \text{ is allocated successfully} \\ -1, & \text{if } \tau_i \text{ is allocated unsuccessfully} \\ u^{\tau_i} - t_0, & \text{if } \tau_i \text{ is sent to the server.} \end{cases} \quad (8)$$

Here, u^{τ_i} is the utility of task τ_i, $t_q^{\tau_i}$ is the number of time steps that task τ_i stays in the queue, and t_0 is a constant representing the number of time steps used to send a task to the server. By the definition of reward r_e, if action a^{τ_i} successfully allocates task τ_i, the reward the edge host receives is the difference between the utility of the task and the number of time steps used to complete the task. We assume $u^{\tau_i} > (t^{\tau_i} + \lceil\tau_i/c^{\tau_i}\rceil)$, otherwise, the edge host would not be motivated to complete any tasks. Conversely, if the task allocation is not successful, the task stays in the queue for the next time step, and action a^{τ_i} results in a penalty of 1. Cases where task τ_i is sent to the server, i.e., the action is $\langle 0, 0\rangle$, are rewarded with $u^{\tau_i} - t_0$ reward. We assume that $u^{\tau_i} > t_0$, otherwise, the edge host would not be motivated to send any tasks to the server. These three levels of reward deter unsuccessful allocations with a negative reward; favor completion by the edge host in the fastest possible time with the highest reward; but still provide the option for the host to delegate a task to the server to avoid a penalty.

Unlike edge hosts, the server's reward is based on: 1) the number of resources reserved for the edge hosts and 2) the number of time steps used to complete a task. Formally, given action $\langle d_1, \ldots, d_{n_e}\rangle$, the reward of the server r_s is defined as

$$r_s = \begin{cases} -\sum_{i=1}^{n_e} d_i, & \text{if no task is received,} \\ u^{\tau_i} - \lceil\frac{\tau_i}{d_i}\rceil - \sum_{i=1}^{n_e} d_i, & \text{if } \tau_i \text{ is received from host } i. \end{cases} \quad (9)$$

The definition of reward r_s states that by performing action $\langle d_1, \ldots, d_{n_e}\rangle$, if no task is received, the server gets a negative reward as its resources are reserved for nothing. When a task is received, its utility is counted into the server's reward. Likewise, in the case that multiple tasks are received simultaneously, their utility is counted into the server's reward. This definition of reward r_s encourages the server to reserve resources only when they are required.

F. Algorithms

The deep reinforcement learning algorithm for each edge host is presented in Algorithm 1. In each time step, an edge host, indexed by i, receives the server's reward signal r_s^i (line 5). The edge host then observes a state s^t (line 6). For a given task τ, an action is selected through ϵ-greedy (line 7). After performing the action, the edge host receives a reward r_e and updates its state s^{t+1} (lines 8 and 9). The states, action, and reward are combined as an experience sample $\langle s^t, a^\tau, r_e, s^{t+1}\rangle$ stored in replay set D (line 10). The edge host then randomly selects a batch of samples B from the replay set D to calculate the loss function $L(\theta)$ and then update the weight θ (lines 11–13). Finally, in line 14, the edge host sends the output of its local DQN, i.e., a Q-vector $\boldsymbol{Q}(s^t)$, to the server.

Algorithm 1: Edge Hosts' Algorithm

```
1  Initialize the weight $\theta$ of the DQN randomly;
2  Initialize the experience replay set $D = \emptyset$;
3  Initialize the $Q$-values of each action randomly;
4  for $t = 1$ to $\infty$ do
5      Receive the server's reward signal $r_s^i$, where $i$ denotes
       the current edge host;
6      Observe the state $s^t$;
7      For a given task $\tau$, with probability $\epsilon$, randomly
       select an action $a^\tau$. Otherwise select
       $a^\tau = argMax_{a^\tau} Q(s^t, a^\tau; \theta)$;
8      Perform action $a^\tau$;
9      Receive the reward $r_e$ and enter a new state $s^{t+1}$;
10     Put sample $\langle s^t, a^\tau, r_e + r_s^i, s^{t+1}\rangle$ into $D$;
11     Randomly select a batch of samples $B$ from $D$;
12     Calculate $L(\theta)$ using Eq. (5);
13     Update the weight $\theta$ using gradient descent;
14     Send the output $Q$-value vector $\boldsymbol{Q}(s^t)$ to the server;
```

Algorithm 2: Server's Algorithm

```
1  Initialize the weight $\theta'$ of the DQN randomly;
2  Initialize the experience replay set $D' = \emptyset$;
3  Initialize the $Q$-values of each action randomly;
4  for $t = 2$ to $\infty$ do
5      Receive the $Q$-value vectors from each edge host;
6      Combine these vectors and form a state $o^t$;
7      Put the sample $(o^{t-1}, a_s^{t-1}, r_s^{t-1}, o^t)$ into $D'$;
8      With probability $\epsilon$, randomly select an action $a_s$.
       Otherwise select $a_s = argMax_{a_s} Q(o^t, a_s; \theta')$;
9      Perform action $a_s$ and receive the reward $r_s$;
10     Randomly select a batch of samples $B'$ from $D'$;
11     Calculate $L'(\theta)$ using Eq. (5);
12     Update the weight $\theta'$ using gradient descent;
13     Divide the reward $r_s$ into $n_e$ pieces and send each
       piece back to a corresponding edge host;
```

Algorithm 3: Division of the Server's Reward

```
1  Input: the server's reward $r_s$;
2  Output: rewards $r_s^1, ..., r_s^{n_e}$;
3  for $i = 1$ to $n_e$ do
4      if Edge host $i$ sends a task $\tau_i$ to the server then
5          $r_s^i \leftarrow u^{\tau_i} - \lceil \frac{\tau_i}{d_i} \rceil - d_i$;
6      else
7          $r_s^i \leftarrow -d_i$;
```

The server will also adopt a deep reinforcement learning algorithm (Algorithm 2) for resource division and will send back a reward signal to the edge host.

Once the server receives Q-value vectors from each edge host, it combines them to form a state o^t, which is in the form of a matrix (lines 5 and 6). The server then creates an experience sample $(o^{t-1}, a_s^{t-1}, r_s^{t-1}, o^t)$ using the state, action, and reward obtained in the last learning round, and stores the sample in replay set D' (line 7). The server cannot observe a new state until in the next learning round when it receives Q-value vectors from edge hosts. The server selects an action a_s, i.e., a resource division strategy, through ϵ-greedy (line 8). After performing the action, the server receives the reward r_s (line 9). The server then randomly selects a batch of samples B' from replay set D' to calculate the loss function $L'(\theta)$ and then update the weight θ' (lines 10–12). Finally, the server divides reward r_s into n_e pieces, where n_e is the number of edge hosts, and sends back each piece to an edge host. The reward r_s is divided using Algorithm 3.

The division of reward r_s is based on the number of resources reserved for each edge host and whether edge hosts send tasks to the server. Specifically, for edge host i, given that the number of reserved resources is d_i, if edge host i sends task τ_i to the server, the reward division for edge host i, r_s^i, is $u^{\tau_i} - \lceil \tau_i/d_i \rceil - d_i$ (line 5); otherwise, the reward division for edge host i is $-d_i$ (line 7). Each reward division is a signal to inform an edge host whether its last action is good or not to the server.

V. Theoretical Analyses

A. Utility Analysis

The CFRL-based learning scheme relies on reward shaping by the server to guide the learning of each edge host, i.e., each edge host's reward is added by part of the server's reward (line 10 in Algorithm 1). Reward shaping is a method of incorporating domain knowledge into reinforcement learning to accelerate learning [42]. One standard method of reward shaping is to alter the rewards of the original process so that the algorithm is able to detect the long-term consequences of its actions more quickly [43].

A typical formulation for a reward shaping function is as follows:

$$F(s, s') = \gamma \Phi(s') - \Phi(s) \tag{10}$$

where Φ is a real-valued function over states. Ng *et al.* [42] proved that by adding an additional reward to the original reward, in the form of (10), i.e., $r + F(s, s')$, the final policy would be guaranteed to be equivalent to the original policy. However, in our problem, the rewards must be shaped based on the actions of both the edge hosts and the server (Algorithm 3). Therefore, the real-valued function Φ would best be defined over actions rather than states. We have, therefore, redefined the reward shaping function as

$$\hat{F}\left(a_i^t, a_s^t, a_i^{t+1}, a_s^{t+1}\right) = \gamma\left(\Phi\left(a_s^{t+1}\right) + \Phi\left(a_i^{t+1}\right)\right) - \left(\Phi\left(a_s^t\right) + \Phi\left(a_i^t\right)\right) \tag{11}$$

where t is the index of time steps, i is the index of edge hosts, a_i^t is the action taken by edge host i at time step t, and a_s^t is the action performed by the server at time step t.

To prove that shaping the reward in the form of (11) guarantees invariance in the accumulated reward, we must prove. This leads to Theorem 1.

Theorem 1: Given the reward shaping function in (11), CFRL-based guarantees invariance in the utility received by the edge host i: $U(\overline{a}_i) = U_\Phi(\overline{a}_i)$, where $\overline{a}_i$ denotes a trajectory consisting of a set of state and action pairs: $\overline{a}_i = \langle (s_i^0, a_i^0); \cdots; (s_i^N, a_i^N) \rangle$.

Proof: Given $U(\overline{a}_i) = \sum_{t=0}^{N-1} \gamma^t r(s_i^t, a_i^t)$ and $U^\Phi(\overline{a}_i) = \sum_{t=0}^{N-1} \gamma^t (r(s_i^t, a_i^t) + \hat{F})$, we have

$$\begin{aligned} U^\Phi(\overline{a}_i) &= \sum_{t=0}^{N-1} \gamma^t \left[r\left(s_i^t, a_i^t\right) + \hat{F} \right] \\ &= \sum_{t=0}^{N-1} \gamma^t \Big[r\left(s_i^t, a_i^t\right) + \gamma\left(\Phi\left(a_s^{t+1}\right) + \Phi\left(a_i^{t+1}\right)\right) \\ &\qquad - \left(\Phi\left(a_s^t\right) + \Phi\left(a_i^t\right)\right) \Big] \\ &= \underbrace{\sum_{t=0}^{N-1} \gamma^t r\left(s_i^t, a_i^t\right)}_{U(\overline{a}_i)} + \sum_{t=0}^{N-1} \gamma^{t+1} \Phi\left(a_s^{t+1}\right) \end{aligned}$$

$$
\begin{aligned}
&+\sum_{t=0}^{N-1}\gamma^{t+1}\Phi\left(a_i^{t+1}\right)\\
&-\sum_{t=0}^{N-1}\gamma^{t}\Phi\left(a_s^{t}\right)-\sum_{t=0}^{N-1}\gamma^{t}\Phi\left(a_i^{t}\right)\\
=&\,U(\overline{a}_i)+\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_s^{t}\right)+\gamma^N\Phi\left(a_s^N\right)+\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_i^{t}\right)\\
&+\gamma^N\Phi\left(a_i^N\right)-\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_s^{t}\right)-\Phi\left(a_s^0\right)\\
&-\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_i^{t}\right)-\Phi\left(a_i^0\right)\\
=&\,U(\overline{a}_i)+\gamma^N\Phi\left(a_s^N\right)+\gamma^N\Phi\left(a_i^N\right)-\Phi\left(a_s^0\right)-\Phi\left(a_i^0\right).
\end{aligned}
$$

To guarantee the invariance in the utility, $U(\overline{a}_i)=U_\Phi(\overline{a}_i)$, we need to ensure $\gamma^N\Phi(a_s^N)+\gamma^N\Phi(a_i^N)-\Phi(a_s^0)-\Phi(a_i^0)=0$.

First, since the server does not give reward signal to any edge host at time step 0, both $\Phi(a_s^0)$ and $\Phi(a_i^0)$ equal to 0. Next, Algorithm 3 stipulates that the server does not give a reward for the last task of each edge host. Hence, both $\gamma^N\Phi(a_s^N)$ and $\gamma^N\Phi(a_i^N)$ equal to 0. Moreover, because i is an arbitrary edge host in the system, this proof can be applied to all the edge hosts. ■

Remark 1: In Algorithm 3, the reward signal sent by the server to an edge host is based on the current actions of both the server and the edge host. However, in (11), the reward shaping function $\hat{F}$ requires that the shaping reward be based on both the current and next actions of both the server and the edge host. This appears to be inconsistent, but it is not. To explain why, we must revisit the summation $\sum_{t=0}^{N-1}\gamma^{t+1}\Phi(a_s^{t+1})+\sum_{t=0}^{N-1}\gamma^{t+1}\Phi(a_i^{t+1})$, which involves the next actions of both the server and edge host i. By splitting $\sum_{t=0}^{N-1}\gamma^{t+1}\Phi(a_s^{t+1})$ and $\sum_{t=0}^{N-1}\gamma^{t+1}\Phi(a_i^{t+1})$, we have

$$
\begin{aligned}
&\sum_{t=0}^{N-1}\gamma^{t+1}\Phi\left(a_s^{t+1}\right)+\sum_{t=0}^{N-1}\gamma^{t+1}\Phi\left(a_i^{t+1}\right)\\
&\quad=\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_s^{t}\right)+\gamma^N\Phi\left(a_s^N\right)+\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_i^{t}\right)+\gamma^N\Phi\left(a_i^N\right).
\end{aligned}
\tag{12}
$$

Equation (12) shows that next actions can be incorporated into current actions by properly setting the index of time steps. Moreover, as discussed in the proof of Theorem 1, both $\gamma^N\Phi(a_s^N)$ and $\gamma^N\Phi(a_i^N)$ equal 0. Therefore, we have

$$
\begin{aligned}
&\sum_{t=0}^{N-1}\gamma^{t+1}\Phi\left(a_s^{t+1}\right)+\sum_{t=0}^{N-1}\gamma^{t+1}\Phi\left(a_i^{t+1}\right)\\
&\quad=\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_s^{t}\right)+\sum_{t=1}^{N-1}\gamma^{t}\Phi\left(a_i^{t}\right)
\end{aligned}
$$

which only includes the current actions of the server and edge hosts, resolving the inconsistency.

As a last point on utility, Theorem 1 indicates that each edge host can still receive its expected utility even if the server intervenes in the task processing. Having a server process a task will increase resource utilization for the server because the server's own rewards are based on the amount of resources it has reserved for the edge hosts and used for itself.

B. Privacy Analysis

As discussed in Section I, existing federated learning-based methods are vulnerable to adversarial activities, while our method can overcome these activities to a large extent. We first analyze why existing methods are vulnerable, and then explain how our method overcomes the vulnerability.

In existing federated learning-based methods, each edge host must share its model parameter updates, i.e., gradients, with the server. The server, however, can reconstruct the input data of each edge host. The reason why the server can do the reconstruction is explained as follows. Training a machine learning model, e.g., a neural network, is equivalent to optimizing the parameters θ of the network using a loss function $\mathcal{L}$ and a set of training data $\mathcal{D}$, which involve data records x_i and the corresponding labels y_i to solve

$$
\mathcal{L}_\theta=\min_\theta\sum_{i=1}^{|\mathcal{D}|}\mathcal{L}_\theta(x_i,y_i) \tag{13}
$$

where i is the index of data records. In federated learning, each edge host shares its gradients $\nabla_\theta\mathcal{L}_\theta(x_i,y_i)$ with the server. The server, then, accumulates these gradients to update the overall weights using the following equation:

$$
\theta^{t+1}=\theta^t-\frac{1}{N}\sum_{j=1}^{N}\nabla_\theta\mathcal{L}_{\theta^t}^j \tag{14}
$$

where t is the index of learning rounds, j is the index of edge hosts, and N denotes the number of edge hosts. The parameters θ^{t+1} are sent back to the individual edge hosts to update their local parameters. As only parameters are shared, the information about data records (x,y) is hidden. However, Geiping *et al.* [16] have shown that the information about data records (x,y) can be recovered from the parameters.[1] Given an unbiased fully connected layer mapping the input x_l to the output, after a nonlinear activation function, e.g., ReLU: $x_{l+1}=max(A_lx_l,0)$ for a matrix A_l of compatible dimensionality. Then, by the chain rule, x_l can be computed using the following expression:

$$
\left(\frac{d\mathcal{L}}{d(x_{l+1})_k}\right)^{-1}\cdot\left(\frac{d\mathcal{L}}{d(A_l)_{i,:}}\right)^T \tag{15}
$$

where k is the index of the dimensionality.

To overcome this vulnerability, in our CFRL-based method, instead of sharing model parameter updates, each edge host shares only their model outputs with the server. However, sharing outputs may incur another vulnerability: model extraction [44], [45], which aims to duplicate the functionality of the target model by stealing the structure, e.g., the weights,

[1]Currently, the quality of most recovered data, shown in [16], is not good enough. However, the work in [16] offered a novel way to recover private data in federated learning settings and the initial results had proved that not sharing data is not secure enough.

of the model. Once obtaining the structure of the model, an adversary can launch the aforementioned input reconstruction attack. For example, a typical model extraction method is based on equation solving [44]. Given a linear regression model f performing binary classification $c = 2$ with parameters $\mathbf{w} \in \mathbb{R}^d$ and $\beta \in \mathbb{R}$, the model f outputs a probability for a given data record x^* as $f(x^*) = \sigma(\mathbf{w} \cdot x^* + \beta)$, where $\sigma(t) = (1/[1 + e^{-t}])$. Hence, the weights $\mathbf{w}$ can be computed by solving the linear equation: $\mathbf{w} = ([\sigma^{-1}(f(x^*)) - \beta]/x^*)$. Once the weights $\mathbf{w}$ are obtained, an adversary can recover any input data record x by solving another linear equation using the corresponding output: $x = ([\sigma^{-1}(f(x)) - \beta]/\mathbf{w})$. This attack method can also be extended to deep neural networks. Taking softmax models for example, the equations use the form: $f_k(x) = ([e^{\mathbf{w}_k \cdot x + \beta_k}]/[\sum_{j=0}^{c-1} e^{\mathbf{w}_j \cdot x + \beta_j}])$. This equation system can be solved by minimizing an appropriate loss function, e.g., the logistic loss. However, these model extraction attacks have a common drawback that an adversary must access the target model. Thus, an intuitive way to defend against these attacks is to disable the access to the target model. In our method, each edge host has a local model, which is physically located on each host and not allowed to be accessed by any others including the server. Therefore, our method can automatically avoid model extraction attacks. In the extreme case that an attacker gets access to the target model somehow, we can adopt private prediction interface [46] to defend against the attacker. Private prediction interface can obfuscate the output of a model to confuse the attacker. Hence, even if the attacker has access to the model, he cannot utilize the output of the model to launch model extraction attacks.

VI. Experiments

A. Setup

To evaluate *CFRL-based*, we developed a Python simulator via PyTorch 1.2.0 with one server and n_e edge hosts. The server had n_r resources, each edge host had m resources, and the sliding window for each edge host contained n time slices. Hence, the resource allocation status matrix $\mathbf{C}$ had a size of $m \times n$.

The experiment design was configured as follows. The set of tasks given to each edge host was dynamically generated following a Poisson distribution with the mean rate of λ. That is, an average of λ tasks was generated for scheduling in each timestep. Each task required τ resources to complete, where τ followed a discrete uniform distribution within a given range.

The evaluation was done in comparison to two other methods: 1) deep reinforcement learning [4], denoted as DRL-based and 2) the optimal integer linear programming algorithm in [47], denoted as IP-based.

DRL-based [4] is the current state of the art in resource allocation. Its definitions of states and actions are similar to *CFRL-based*'s. However, *DRL-based* does not consider the server and only considers one edge host. By comparison, *CFRL-based* takes multiple edge hosts into account, asking them to learn alongside the server in a federated manner. The difference in performance between *DRL-based* and *CFRL-based*, therefore, reflects the benefits of federated learning and concurrency, as well as providing a comparison to a local method.

IP-based [47] is a broadly used optimization method that has been applied to resource allocation in edge computing environments, albeit without much success. This method is a centralized one that directly consolidates information provided by each edge hosts to optimally allocate resources. It is neither efficient nor applicable to complex edge-computing systems, but it can serve as a benchmark for a global method.

The four evaluation metrics used to assess performance were as follows.

1) *Average utility*, which measures the reward received by each edge host for completing tasks.
2) *Average time step*, which is the average number of time steps used to complete a task.
3) *Average resource utilization*, which is the ratio between the number of utilized resources and the number of allocated resources for each task.
4) *Average communication overhead*, which measures the average number of tasks sent by each edge host to the server.[2]

B. Results

The experiments were conducted in three scenarios. In the first scenario, we varied the number of edge hosts n_e from 2 to 4, while fixing the number of tasks λ at 8 and drawing the number of resources required by each task τ uniformly from the range $[3, 5]$. In the second scenario, we varied λ from 6 to 14 while fixing n_e to 3 and uniformly drawing τ from the range $[3, 5]$. In the third scenario, we varied the range of τ by $[3, 5]$, $[4, 6]$, and $[5, 7]$. n_e was fixed at 3 and λ was fixed at 8. For all three scenarios, each host had ten resources, the server had 30, and the sliding window length was set to 5.

Fig. 5 shows the results for all three scenarios. In every situation, *CFRL-based* had a higher resource utilization rate than *DRL-based* and all tasks took less time to complete. Moreover, the *CFRL-based* method required less communication overheads than *DRL-based* method. We attribute this result to the jointly decision-making process over optimizing resource allocation. The *DRL-based* method optimizes resources at the individual edge host level; it does not take the server or the edge hosts as a group into consideration.

These results provide three further interesting observations. First, as shown in Fig. 5(c), the difference in resource utilization between the two methods decreased as the number of edge hosts increased. Increasing the number of hosts obviously increases the amount of available resources, which provides more flexibility for arranging the tasks in a way that maximizes resource use. This is of help to any resource allocation method.

Second, Fig. 5(d) shows that with the number of edge hosts increasing, the communication overheads in both the *CFRL-based* and *DRL-based* methods remain almost unchanged. This

[2]Actually, in addition to sending tasks, edge hosts and the server also exchange other information, including edge hosts' policies and the server's rewards. Such information exchange, however, is inevitable in all the federated learning-based methods. Thus, it is not counted in our experiments.

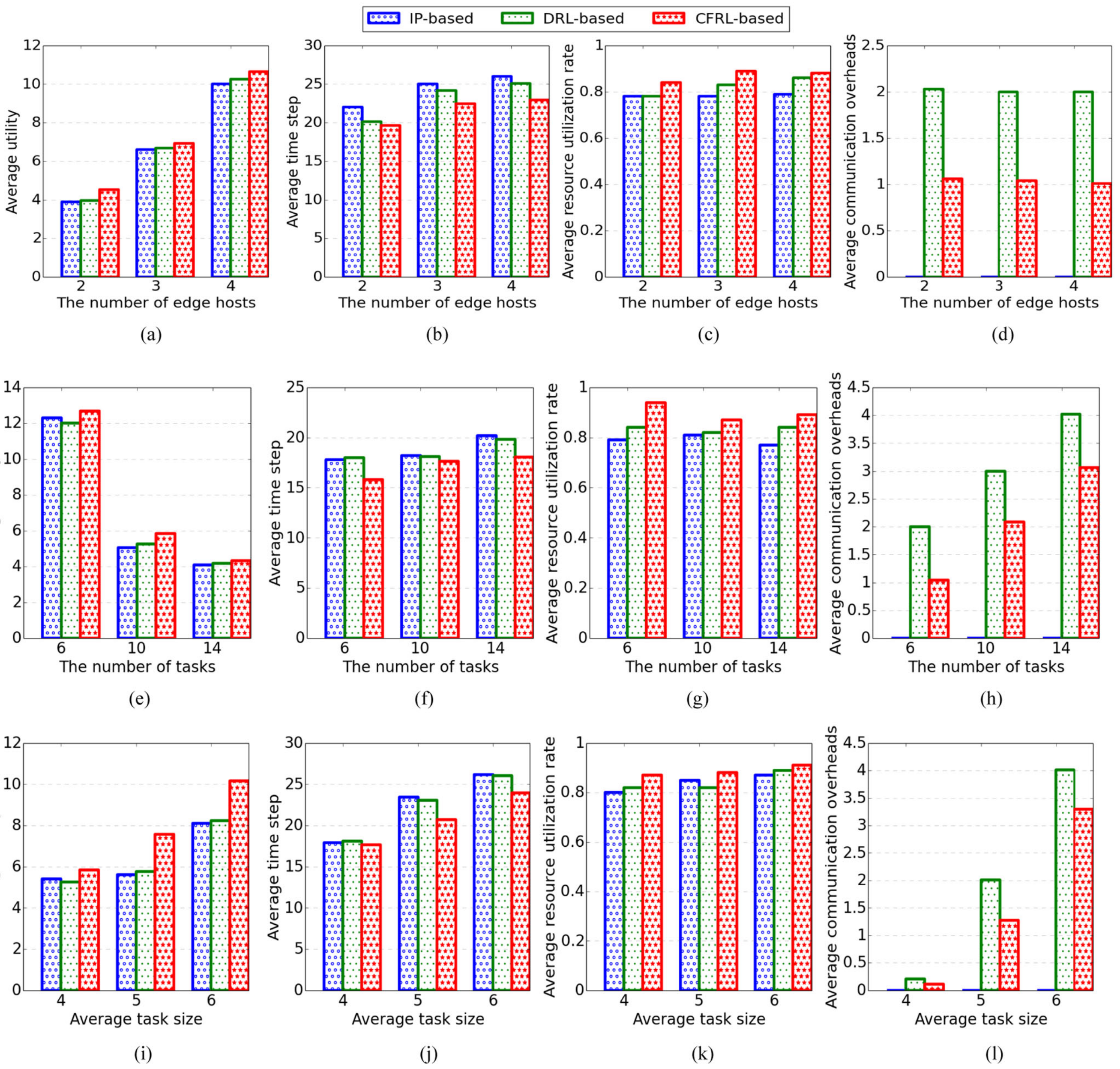

Fig. 5. Performance comparison among our *CFRL-based* method, *IP-based* method, and the *DRL-based* method. (a) Utility with different numbers of edge hosts. (b) Time step with different numbers of edge hosts. (c) Resource utilization rate with different numbers of edge hosts. (d) Communication overheads with different numbers of edge hosts. (e) Utility with different numbers of tasks. (f) Time step with different numbers of tasks. (g) Resource utilization rate with different numbers of tasks. (h) Communication overheads with different numbers of tasks. (i) Utility with different task sizes. (j) Time step with different task sizes. (k) Resource utilization rate with different task sizes. (l) Communication overheads with different task sizes.

is because in both of the two methods, edge hosts do not interact between one another. Thus, each edge host's decision on sending tasks to the server is independent of other edge hosts.

Third, in Fig. 5(i), with the increase of task size, the average utility difference between the *CFRL-based* and *DRL-based* methods is enlarged. This is because a larger task size increases the competition for resources. *CFRL-based* is based in an egalitarian philosophy of jointly generating resource allocation strategies for the benefit of the whole system, whereas the *DRL-based* mindset in competitive environments is more like one of survival by the fittest. Any task that falls by the wayside will reduce overall utilization.

Since the deep *Q*-learning framework in our method needs a training process, Fig. 6 is used to complement the results of Fig. 5 by showing the performance change of the three methods as the training progresses. Here, Fig. 6(b), (f), and (j) shows that the edge hosts took more time to complete their tasks with the *CFRL-based* method than with *DRL-based* in the early stages of learning. This is because, initially, neither the server nor the hosts have much knowledge. When multiple parties jointly learn, one party can negatively affect the others, resulting in high time consumption. In contrast, with the *DRL-based* method, each party learns independently, so the negative influence of others is not an issue. However, as learning progresses, each party accumulates knowledge, and the ability to

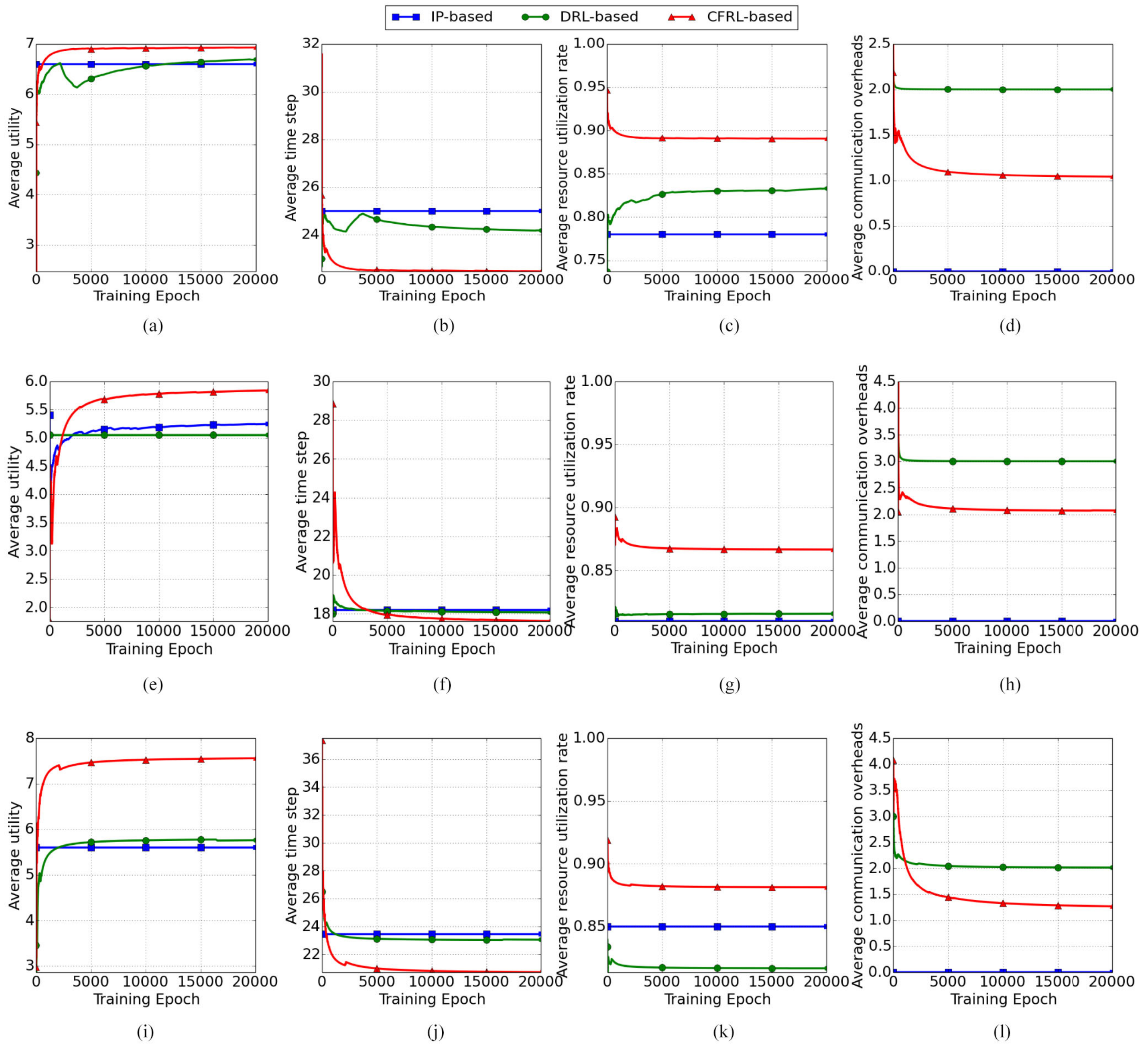

Fig. 6. Performance comparison among our *CFRL-based* method, *IP-based* method, and the *DRL-based* method. (a) Utility as learning progresses with three edge hosts. (b) Time step as learning progresses with three edge hosts. (c) Resource utilization rate as learning progresses with three edge hosts. (d) Communication overheads rate as learning progresses with three edge hosts. (e) Utility as learning progresses with lambda = 10. (f) Time step as learning progresses with lambda = 10. (g) Resource utilization rate as learning progresses with lambda = 10. (h) Communication overheads rate as learning progresses with lambda = 10. (i) Utility as learning progresses with tau drawn from [4, 6]. (j) Time step as learning progresses with tau drawn from [4, 6]. (k) Resource utilization rate as learning progresses with tau drawn from [4, 6]. (l) Communication overheads as learning progresses with tau drawn from [4, 6].

share this knowledge has a very positive impact. Thus, in the later stages of learning, most tasks took less time to complete with *CFRL-based*.

Furthermore, Fig. 6(c), (g), and (k) shows *CFRL-based* managed to achieve very high resource utilization in the early stages, which fell sharply in the later stages. One possible explanation for this phenomena is that, in the early stages, each party allocates less resources to tasks than in the late stages–perhaps in a "test-the-water" kind of strategy. Allocating less resources implies a higher resource utilization but incurs a higher time cost and a lower utility. Since each party's objective is to increase their utility, each party may be inclined to allocate more resources to tasks as learning progresses and they become familiar with which allocation strategies work the best.

VII. Conclusion

In this article, we presented a novel algorithm for resource allocation based on CFRL-based. The method considers the global speed and efficiency of task completion while preserving privacy for both the server and the edge hosts to the greatest possible extent. Compared to the state of the art in deep reinforcement learning, our algorithm gives edge hosts more utility and, system wide, tasks get completed in less time. In the future, we plan to introduce a formal privacy-preserving mechanism into our method to provide a theoretical guarantee of privacy protection for edge hosts.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[2] C. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surveys*, vol. 52, no. 5, pp. 1–37, 2019.

[3] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2061–2073, Apr. 2019.

[4] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020.

[5] A. Alwarafy, K. A. Al-Thelaya, M. Abdallah, J. Schneider, and M. Hamdi, "A survey on security and privacy issues in edge computing-assisted Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4004–4022, Mar. 2021, doi: 10.1109/JIOT.2020.3015432.

[6] J. Konecnc, H. B. McMahan, F. X. Yu, P. Richtarik, A. Theertha, and S. D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop Private Multiparty Mach. Learn.*, 2016, pp. 1–9.

[7] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[8] P. Bellavista, L. Foschini, and A. Mora, "Decentralised learning in federated deployment environments: A system-level survey," *ACM Comput. Surveys*, vol. 54, no. 1, pp. 1–38, 2021.

[9] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, Jan. 2016.

[10] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4555–4562, Oct. 2019.

[11] H. Zhou, W. Feng, Y. Lin, and Q. Yang. (2020). *Federated Deep Reinforcement Learning*. [Online]. Available: https://arxiv.org/abs/1901.08277

[12] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[13] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multi-timescale resource management for multi-access edge computing in 5G ultra dense network," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2238–2251, Feb. 2021, doi: 10.1109/JIOT.2020.3026589.

[14] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2020, pp. 234–243.

[15] X. Wang, R. Li, C. Wang, X. Li, T. Taleb, and V. C. M. Leung, "Attention-weighted federated deep reinforcement learning for device-to-device assisted heterogeneous collaborative edge caching," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 154–169, Jan. 2021.

[16] J. Geiping, H. Bauermeister, H. Droge, and M. Moeller, "Inverting gradients—How easy is it to break privacy in federated learning?" in *Proc. NIPS*, 2020, pp. 157–168.

[17] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen, "Deep reinforcement learning for autonomous Internet of Things: Model, applications and challenges," *IEEE Commun. Survey Tuts.*, vol. 22, no. 3, pp. 1722–1760, 3rd Quart., 2020.

[18] 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.

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

[20] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in IoT networks via reinforcement learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.

[21] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in IoT networks via machine learning," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3415–3426, Apr. 2020.

[22] X. Liu, J. Yu, Z. Feng, and Y. Gao, "Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing," *China Commun.*, vol. 17, no. 9, pp. 220–236, 2020.

[23] K. Zheng, H. Meng, P. Chatzimisios, L. Lei, and X. Shen, "An SMDP-based resource allocation in vehicular cloud computing systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7920–7928, Dec. 2015.

[24] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2018, pp. 1–6.

[25] A. Nassar and Y. Yilmaz, "Resource allocation in fog RAN for heterogeneous IoT environments based on reinforcement learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.

[26] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 10–17, 2019.

[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] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, early access, Mar. 4, 2021, doi: 10.1109/TETC.2019.2902661.

[29] L. Huang, S. Bi, and Y. 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.

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

[31] I. Alqerm and J. Pan, "Enhanced online $Q-$learning scheme for resource allocation with maximum utility and fairness in edge-IoT networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 3074–3086, Oct.–Dec. 2020, doi: 10.1109/TNSE.2020.3015689.

[32] H. Moulin, *Fair Division and Collective Welfare*. Cambridge, MA, USA: MIT Press, 2003.

[33] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported Internet of Things," *IEEE Access*, vol. 7, pp. 69194–69201, 2019.

[34] Y. Qian, L. Hu, J. Chen, X. Guan, M. M. Hassan, and A. Alelaiwi, "Privacy-aware service placement for mobile edge computing via federated learning," *Inf. Sci.*, vol. 505, pp. 6178–6186, Dec. 2019.

[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[36] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," in *Proc. NIPS Deep Learn. Workshop*, 2013, pp. 1–6.

[37] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, vol. 54, 2017, pp. 1273–1282.

[38] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, Apr. 2020, doi: 10.1109/TIFS.2020.2988575.

[39] D. Wu, M. Pan, Z. Xu, Y. Zhang, and Z. Han, "Towards efficient secure aggregation for model update in federated learning," in *Proc. IEEE Global Commun. Conf.*, 2020, pp. 1–6.

[40] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu, and X. Zheng, "Privacy-preserving federated learning framework based on chained secure multiparty computing," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6178–6186, Apr. 2021.

[41] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *Proc. USENIX Security Symp.*, 2019, pp. 1895–1912.

[42] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proc. ICML*, 1999, pp. 278–287.

[43] M. Grzes, "Reward shaping in episodic reinforcement learning," in *Proc. AAMAS*, 2017, pp. 565–573.

[44] F. Tramer, F. Zhang, and A. Juels, "Stealing machine learning models via prediction APIs," in *Proc. USENIX Security Symp.*, 2016, pp. 601–618.

[45] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *Proc. USENIX Security Symp.*, 2020, pp. 1345–1362.

[46] C. Dwork and V. Feldman, "Privacy-preserving prediction," in *Proc. COLT*, 2018, pp. 1–10.

[47] D. Li, Y. Jin, and H. Liu, "Resource allocation strategy of edge systems based on task priority and an optimal integer linear programming algorithm," *Symmetry*, vol. 12, no. 6, pp. 1–19, 2020.

Zhu Tianqing received the B.Eng. and M.Eng. degrees from Wuhan University, Wuhan, China, in 2000 and 2004, respectively, and the Ph.D. degree in computer science from Deakin University, Geelong, VIC, Australia, in 2014.

She is currently a Professor with China University Geosciences, Wuhan. Prior to that, she was a Lecturer with the School of Information Technology, Deakin University, from 2014 to 2018. Her research interests include privacy preserving, data mining, and cyber security.

Wei Zhou received the Bachelor of Science degree in electrical engineering from Changchun University of Science and Technology, Changchun, China, in 2020. He is currently pursuing the first-year graduate degree in computer science with China University of Geosciences, Wuhan, China.

His research interests include reinforcement learning and differential privacy.

Dayong Ye received the M.Sc. and Ph.D. degrees from the University of Wollongong, Wollongong, NSW, Australia, in 2009 and 2013, respectively.

He is currently a Research Fellow of Cybersecurity with the University of Technology Sydney, Sydney, NSW, Australia. His research interests focus on differential privacy, privacy preservation, and multiagent systems.

Zishuo Cheng received the B.I.T. and M.Comp. degrees from Australian National University, Canberra, ACT, Australia, in 2013 and 2015, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science, University of Technology Sydney, Sydney, NSW, Australia.

His research interests include privacy preservation, multiagent systems, and machine learning.

Jin Li (Senior Member, IEEE) received the B.S. degree in mathematics from Southwest University, Chongqing, China, in 2002, and the Ph.D. degree in information security from Sun Yat-sen University, Guangzhou, China, in 2007.

He is currently a Professor with Guangzhou University, Guangzhou. He has authored or coauthored more than 50 research papers in refereed conferences and journals and has been the Program Chair or the Program Committee Member for many international conferences. He has been also named one of Guangdong Province's 'New Stars in Science and Technology. His research interests include applied cryptography and security in cloud computing.