FedTO: Mobile-Aware Task Offloading in Multi-Base Station Collaborative MEC

Zhao Tong, *Member, IEEE*, Jiake Wang, Jing Mei, Kenli Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—With the proliferation of the Internet of Things (IoT), mobile edge computing (MEC) has great potential to achieve low latency, high reliability, and low energy consumption. However, in collaborative MEC environments, user movement and task migration may cause task transmission and processing delays, resulting in elevated task response times. Therefore, system performance and user experience need to be ensured by rational task offloading and resource management. At the same time, the protection of user data privacy is becoming increasingly important as a challenge to be overcome. To address the problems of intense resource competition and privacy leakage in MEC, the federated learning for the TD3-based task offloading (FedTO) algorithm is proposed. The algorithm has a dual objective of energy consumption and task response time while protecting user privacy. It employs a cryptographic local model update and aggregation mechanism and uses deep reinforcement learning (DRL) to obtain an efficient task offloading decision. Based on the mobile trajectories of real devices, and the pre-deployment of base station locations, experimental results show that the FedTO algorithm ensures task data security. It also effectively reduces the total energy consumption and average task response time of the system, which further improves the system utility.**

***Index Terms*—Federated deep reinforcement learning, mobile edge computing, multiple base stations collaboration, task offloading, user mobility.**

I. Introduction

At PRESENT, with the rapid development and convergence of the Internet of Things (IoT), 5G and artificial intelligence technologies, the number of mobile devices and the demand for applications are growing [1]. According to the IoT analytics report, the number of connected IoT devices worldwide will grow by 18% to reach 14.4 billion in 2022 and is expected to reach 27 billion in 2025 [2]. The excessive number of mobile device connections will lead to network congestion, affecting data transmission speed and quality. Additionally, this approach increases the energy demands on the network infrastructure [3]. In addition, the dramatic increase in mobile devices will lead to increased security issues, such as devices leaking user information or devices being hacked [4]. Mobile edge computing (MEC) is a promising technology that provides high-quality services to mobile users [5], [6], [7]. In MEC systems, compute-intensive tasks can be offloaded from mobile devices to nearby edge servers, leading to improved service quality for users and reduced energy consumption of mobile devices. A key challenge for MEC systems is task offloading, the process of deciding which tasks should be offloaded to edge servers and which tasks should be processed locally on mobile devices. In addition, multiple base stations (BSs) can work together to provide computing and storage resources to users. This allows users to access the required computing and storage resources more quickly, while also reducing communication latency and energy consumption [8]. However, task offloading in multiple BSs collaboration is a more complex optimization problem involving trade-offs between energy consumption, processing time and privacy for mobile users. One of the key issues for IoT and MEC systems is to provide high-quality services to end-users while minimizing energy consumption and reducing response times. With the widespread adoption of IoT devices and the increase in real-time demand, energy consumption and latency are growing exponentially. Therefore, effective task offloading strategies are necessary to optimize the performance of IoT and MEC systems. Another major issue is user privacy. On the one hand, data generated by IoT devices may contain sensitive personal information that needs to be protected from unauthorized access. On the other hand, MEC systems need to share data among different parties for efficient computation and communication, while also ensuring privacy protection. Therefore, it is crucial to design privacy protection mechanisms in IoT and MEC.

Traditional task offloading algorithms may not be able to fully handle these problems, while the federated learning (FL) combined with deep reinforcement learning (DRL) can optimize the system utility by minimizing both energy consumption and task response time, without sacrificing user privacy [9], [10]. Specifically, FL protects user privacy by training the model

Manuscript received 26 May 2023; revised 6 October 2023; accepted 29 October 2023. Date of publication 1 November 2023; date of current version 14 March 2024. This work was supported in part by the Program of National Natural Science Foundation of China under Grants 62072174 and 62372172, and in part by the Distinguished Youth Science Foundation of Hunan Province, China, under Grant 2023JJ10030. The review of this article was coordinated by Prof. Liehuang Zhu. *(Corresponding author: Zhao Tong.)*

Zhao Tong, Jiake Wang, and Jing Mei are with the College of Information Science and Engineering, Hunan Normal University in Changsha, Hunan 410081, China (e-mail: tongzhao@hunnu.edu.cn; 202170293854@hunnu.edu.cn; jingmei1988@163.com).

Kenli Li is with the College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, Hunan 410082, China (e-mail: lkl@hnu.edu.cn).

Keqin Li is with the College of Information Science and Engineering, Hunan University, Hunan 410082, China, also with the National Supercomputing Center in Changsha, Hunan 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TVT.2023.3329146

locally on the device and spreading user data and model parameters across multiple devices. The model aggregation process in the FL algorithm involves only the model parameters and not the raw data. Therefore, the user's task data is retained on the local device and is not directly exposed to the central server or other devices. In addition, FL further enhances user privacy protection using encryption techniques [11]. Homomorphic encryption allows computations to be performed in an encrypted state without decrypting the data. Differential privacy uses the method of adding random noise to the data. DRL can generate efficient and adaptive task offloading decisions. First, DRL is able to learn and optimize complex nonlinear functions, making it widely used in multiple BSs collaborative MEC systems to help solve complex optimization problems. Second, due to the uncertainty of mobile users' trajectories and task requirements, traditional optimization algorithms are difficult to find optimal solutions. In contrast, DRL can adaptively adjust its strategy to adapt to the changing environmental state. This makes DRL a well-suited algorithm for solving task offloading problems in dynamic environments. Finally, DRL can handle high-dimensional data with a large number of state spaces [12].

In this paper, the FL framework and twin delayed deep deterministic policy gradient (TD3) algorithm in DRL are combined to optimize the system utility, reduce energy consumption and response time based on safeguarding user privacy. Solve the problem of collaborative task offloading based on mobile users with multiple BSs in the MEC environment and optimize it based on the real-time location information of mobile devices. Therefore, the federated learning for the TD3-based task offloading (FedTO) algorithm is proposed. The following are the main contributions of this paper.

1) We propose a multiple BSs collaborative MEC scenario with heterogeneous computing power. In this scenario, a Markov decision process (MDP) considering user mobility and task migration cost is established. Bi-objective optimization is achieved for multiple types of tasks, including energy consumption and task response time.
2) We combine the FL framework and TD3 algorithm to protect user data privacy using encrypted local model update and aggregation mechanisms. Based on the dynamically changing network state and service requirements, an efficient FedTO algorithm is proposed to provide users with a high-quality service experience.
3) We compare four different DRL algorithms based on a real mobile device trajectory dataset in FL framework. BSs are pre-deployed based on the k-means algorithm before making decisions. For task density, size, distribution, and bandwidth scaling, the FedTO algorithm is able to optimize the system utility, reduce energy consumption and task response time under different conditions.

This paper is structured as follows. Section II presents the related work. In Section III, the system model and problem formulation are introduced. The FedTO algorithm is detailed in Section IV. The proposed algorithm's performance is analyzed through experimental evaluation in Section V. Finally, the paper concludes in Section VI.

II. Related Work

A. Task Offloading in Mobile Edge Computing

To improve computational performance and reduce the latency of data processing, task offloading is widely applied in MEC [13], [14], [15]. Wu et al. [16] presented an high-efficiency offloading algorithm that combines MEC and mobile cloud computingin a blockchain scenario, which can reduce system delay and energy while ensuring data integrity. Tan et al. [6] considered the problem of resource allocation in multi-user MEC network and proposes a two-level framework based on heuristic algorithm and DRL method. The proposed solutions show high energy efficiency in different parameter environments. Sun et al. [5] introduced an offloading method predicated on predicting resource occupancy for a resource-poor offloading scenario. The method uses a controlled recursive unit (GRU) to predict the resource utilization of the server and uses the RL algorithm to develop an optimal policy for task offloading. However, most of the models considered in the above works are single base station, and the multiple base stations collaborative environments are more complex and reasonable task offloading decisions are more important. Moreover, these studies assume that mobile devices are stationary and do not consider the mobility of mobile users, which may affect the performance of task offloading. In addition, most models do not consider the delay of task result return and the migration cost.

B. Artificial Intelligence in Mobile Edge Computing

Artificial intelligence technologies enable the optimization of task offloading decisions in an intelligent way, making the system achieve optimal performance and resource utilization efficiency, thus satisfying users' needs in terms of latency, bandwidth, etc [17], [18], [19]. To overcome the challenges of limited sample types and high exploration expenses in MEC, Yao et al. [20] introduced a distributed DRL-based offloading algorithm for function-as-a-service (FaaS). The proposed method accelerates the convergence of the DRL model, as demonstrated by the experimental results. Zhou et al. [21] optimized for deep neural network (DNN) model by transforming a DNN layer into several smaller layers. Moreover, a tradeoff between the efficiency of task offloading and the overhead of model parallelism is made to optimize the model fusion and offloading strategies. A particle swarm optimization algorithm with reduced latency is proposed. Tang et al. [10] modeled the binary task offloading problem for latency-sensitive tasks by optimizing the system cost based on the dynamic load of edge servers, and proposed a DRL-based algorithm. The algorithm effectively reduces task discard rate and task latency and improves the quality of user experience compared with the remaining three benchmarks. However, it is important to protect user privacy in the MEC environment. This is because inadequately protected mission data information can be accessed, leaked or misused maliciously, leading to undesirable consequences. None of the above studies addressed this issue.

TABLE I
COMPARISON OF REFERENCES ABOUT TASK OFFLOADING

Reference	Environment(s)	Objective(s)	Task type	Main technique(s)	Privacy
[16]	MEC, Mobile Cloud Computing	latency, energy	single	Lyapunov	yes
[17]	MEC	energy	single	Ant Colony Algorithm, DQN	no
[18]	Cloud-edge Collaboration	latency, task failure ratio	single	DDPG	no
[22]	Edge Computing	latency	single	Actor-Critc	no
[23]	MEC	DNN inference time	single	PSO	yes
[24]	MEC, Fog Computing	latency, dropped tasks ratio	two	model-free DRL Algorithm	no
[28]	MEC	latency	single	K-neighbor	yes
[29]	MEC	latency, energy	single	DQN	yes
[30]	Cloud-Edge Collaboration	latency, utility	single	Genetic Algorithm	yes
This work	MEC with heterogeneous servers	latency, energy, utility	four	TD3	yes

C. Privacy Protection in Mobile Edge Computing

The tasks being processed contain private information of the company or individual users, such as location, health and behavior. Therefore, how to adopt various privacy protection techniques to safeguard users' privacy in the MEC environment is now widely studied [22], [23], [24]. Wang et al. [25] introduced a local differential privacy algorithm to secure the vehicle user when offloading tasks. In addition, a k-neighbor algorithm is used to generate task offloading decisions to reduce the latency of executing tasks. Zhang et al. [26] considered the unreliability of edge servers and weighed the offloading cost against the privacy level to formulate it as a joint optimization problem. The best offloading solution is obtained using deep Q-network (DQN) for the purpose of improving task privacy level and reducing system energy consumption and latency. Xu et al. [27] proposed a two-stage offloading algorithm with the goal of reducing privacy leakage of edge computing units while optimizing resource utilization and time. In the first phase, with the goal of optimizing the utilization of user resources and minimizing time delays, the algorithm is based on a utility-aware design. In the second phase, a joint optimization approach is proposed to trade-off privacy protection and performance. Moreover, the feasibility of the algorithm is demonstrated by simulation experiments. However, the above studies use traditional privacy policies still require centralizing data to servers for processing, and there is a risk of data leakage and privacy exposure during transmission. In addition, in many scenarios, data are distributed among multiple devices and cannot be stored and processed centrally.

In this paper, the task execution migration cost of mobile users is additionally considered based on a heterogeneous MEC environment with multi-user and multi-base station collaboration. The k-means method is applied for pre-deployment of all base stations before the task offloading decision is generated. In addition, a mobile trajectory-aware task offloading algorithm is proposed. The algorithm is based on the FL framework with distributed training of DRL models for all local agents and central aggregation. Transferring models rather than task data reduces the threat of privacy breaches. In addition, it improves system utility, reduces energy consumption and latency, and ensures user service experience. The reference comparisons are summarized in Table I.

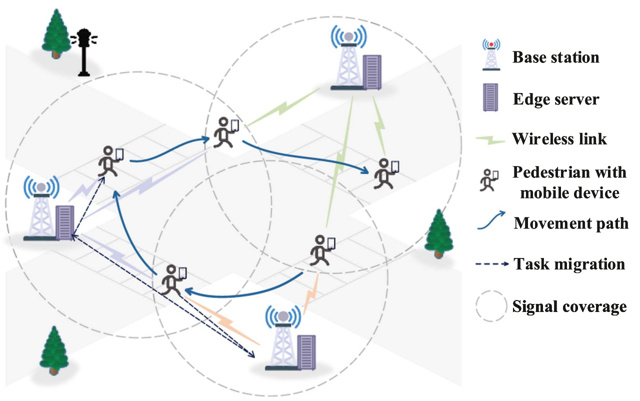


Fig. 1. Task offloading model in a collaborative multi-base station MEC environment.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, An MEC model for collaborative heterogeneous computing capabilities of multi-BSs with two layers is proposed and formulate the problems to be solved in the model.

In this study, a two-tier MEC architecture is considered, consisting mainly of end-user and edge layers, as shown in Fig. 1. The end-user layer comprises $\mathrm{K} = \{1, 2, \ldots, K\}$ mobile devices that receive services. At time slot t, it is assumed that device k generates a task. The types of tasks generated show differences due to the diversity of performance levels and application requirements of individual devices. This model primarily considers four types of tasks, namely text, image, audio, and video. The frequency of local computation varies for each mobile device, while the speed of movement and the trajectory of movement also show diversity. The local computing frequency of a device determines the time it takes to perform a task locally, while the mobility speed and trajectory determine the frequency and cost of switching the device between different base stations

TABLE II
KEY SYMBOLS IN THE MODEL

Notion	Definition
$d_{k,b}$	The distance between device k and BS b
$h_{k,b}$	The channel gain between device k and BS b
$r^r_{k,b}$	The transmission rate between device k and BS b
o_k	The coordinates of the location of device k when the task is submitted
$o_k{}'$	The coordinates of the location of device k when the task is completed
o_b	The coordinates of BS b
$E^{comp}_{k,s}$	The energy consumption of task computation
$E^{trans}_{k,b}$	The total energy consumption of task transmission
$E^{ttl}_{k,b}$	The total energy consumption of the task
M_k	The task generated by mobile device k
R_b	The radius of the BS b wireless signal
$T^{comp}_{k,b}$	The delay of task computation
T^l_k	The delay of local computation in device k
$T^{trans}_{k,b}$	The total delay of task transmission
$T^{ttl}_{k,b}$	The total delay of the task generated by device k
$T^m_{b,w}$	The migration delay from BS b to w
U_k	The system utility of the task M_k
∂^{ty}_k	The type of the task M_k
∂^u_k	The uplink size of the task M_k
∂^r_k	The downlink size of the task M_k
∂^{su}_k	The submission time of the task M_k
∂^{de}_k	The deadline of the task M_k
ξ^b_k	The identifier of the task offloading location

or edge nodes. The offloading strategy needs to fully consider the unique characteristics of each device to ensure the maximization of system efficiency and user experience. The edge layer consists of B BSs that are equipped with servers that provide computing, storage, and network resources. Wireless transmission is used for communication between the mobile device and the BS, and wired transmission is used for communication between the BSs. Where wireless transmission exists in two access modes, denoted using $A_k \in \{0, 1\}$, where 0 denotes cellular data access, and 1 denotes Wi-Fi data access. The radio frequency radius and transmission network bandwidth of BSs are different for different access modes. After a task is generated from a device at the end-user layer, the corresponding environment, and the task information are transmitted to the decision network. The coverage areas of multiple BSs in the model overlap, requiring consideration of factors such as the user's current location, task status, and server status to obtain the most suitable computing node for offloading. If the mobile device is located within the radio coverage of multiple BSs, then all the BSs can serve the device, and the device can choose one of the computing nodes for offloading. After the task is completed, the migration process of the task results is additionally considered in the model due to the mobility of the device. Due to the performance variation of different processors for heterogeneous tasks, the server integrates three types of processor chip resources, namely CPU, GPU, and FPGA [28]. In this model, it is assumed that the MEC BS server can simultaneously receive data, perform task computation, and offload tasks. To facilitate comprehension, Table II presents the key symbols of this model.

A. User Mobility Model

The MEC model location is modeled as a planar Cartesian coordinate system. B BSs are distributed in the plane as specified, and the coordinate of BS b are mapped to $o_b = (x_b, y_b)$. At time t, the mobile device's coordinates are denoted by $o_k(t) = (x_k(t), y_k(t))$. In the two-dimensional plane, the size of the mobile device distance from the BS is given as $d_{k,b}(t) = \|o_k(t) - o_b\|$, where $\|\cdot\|$ denotes the Euclidean parametrization, that is,

$$d_{k,b}(t) = \sqrt{(x_k(t) - x_b)^2 + (y_k(t) - y_b)^2}. \tag{1}$$

After the task is calculated, the mobile device coordinate are represented by $o_k{}' = o_k(t + T^{ttl}_{k,b})$, where $T^{ttl}_{k,b}$ denotes the total response time of the task.

B. Task Model

In the MEC model, a task generated by device k in time t is denoted as $M_k(t) = \{\partial^{ty}_k(t), \partial^u_k(t), \partial^r_k(t), \partial^{su}_k(t), \partial^{de}_k(t)\}$, where $\partial^{ty}_k(t)$ represents the type of task, $\partial^{ty}_k(t) = 0$ for picture task, $\partial^{ty}_k(t) = 1$ for audio task, $\partial^{ty}_k(t) = 2$ for video task, and $\partial^{ty}_k(t) = 3$ for text task. Different types of tasks have different sizes, and heterogeneous processor chips have different processing speeds for various tasks. $\partial^u_k(t)$ means the upload task size. $\partial^r_k(t)$ is the size of the task's computed result. $\partial^{su}_k(t)$ denotes the task submission time, and $\partial^{de}_k(t)$ indicates the deadline of the task. The model uses a binary offload approach where tasks cannot be split. The task can be processed locally or sent to the edge, which can be represented as $\xi^b_k(t) \in \{0, 1\}$. If $\xi^b_k(t) = 0$, the task is computed locally; otherwise, it is offloaded to the edge. Each task is only offloaded to one computing node, denoted as $\sum_{i=0}^{B} \xi^i_k(t) = 1$.

C. Task Offloading Model

The model consists of two parts, i.e., the computation model and the communication model. Among them, the computation model includes local computation and edge-side computation. If the task is offloaded for edge-side computation, communication is necessary. The communication model involves two processes, namely task uplink and task downlink.

1) Computation Model: Tasks are generated locally. It can be executed locally or at edge computing nodes.

a) Local Computing: When $\xi^b_k(t) = 0$, use local arithmetic to process the task, and the delay of the task is determined by the local computation delay. The local computation frequency is denoted by f_k, and c^l_k denotes the number of CPU cycles required by the mobile device k to compute 1-bit task locally. Therefore, the delay $T^l_k(t)$ for the task to compute locally can be expressed as

$$T^l_k(t) = \frac{\partial^u_k(t) c^l_k}{f_k}. \tag{2}$$

The device performs 1 CPU cycle and consumes energy as $\kappa(f_k)^2$, where κ denotes the effective power switch in the chip [29]. Therefore, the energy consumed by the task M_k for computation at device k is denoted as

$$E_k^l(t) = \kappa(f_k)^2 \partial_k^u(t) c_k^l. \tag{3}$$

b) Offloading to the edge: When mobile device k enters the coverage area of BS b, it can request to offload its task to the edge server on the BS. After uploading the task to the BS, the computation can be started by waiting for the server to be idle. The computing delay of the task $T_{k,b}^e(t)$ can be expressed as

$$T_{k,b}^e(t) = \frac{\partial_k^u(t) c_b^{ty}}{f_b^{ty}(C_b(t))}, \tag{4}$$

where $f_b^{ty}(C_b(t))$ denotes the frequency of different chip processors corresponding to different tasks, and 1 bit of the task requires c_b^{ty} processor cycles for computation. One task can only be placed on one processor for execution and cannot be split, therefore, $C_b(t) \in \{C_c, C_g, C_f\}$, where C_c, C_g, and C_f denote the processor CPU, GPU, and FPGA, respectively. The waiting time of the task $T_{k,b}^w(t)$ is the sum of the computation time of the currently non-executed tasks of the server, which can be expressed as

$$T_{k,b}^w(t) = \sum_{i \in U} T_{i,b}^e, \tag{5}$$

where U is the set of tasks waiting to be computed by this server.

The computation energy consumed by the task offloading to the edge can be expressed as

$$E_{k,b}^e(t) = \partial_k^u(t)\varepsilon, \tag{6}$$

where ε is the energy consumed to compute 1 bit of data.

Therefore, the total computation delay of the task $T_{k,b}^{comp}(t)$ can be expressed as

$$T_{k,b}^{comp}(t) = \begin{cases} T_k^l(t), \xi_k(t) = 0 \\ T_{k,b}^w(t) + T_{k,b}^e(t), \xi_k(t) = 1. \end{cases} \tag{7}$$

The total computation energy consumption of the task $E_{k,b}^{comp}(t)$ can be expressed as

$$E_{k,b}^{comp}(t) = \begin{cases} E_k^l(t), \xi_k(t) = 0 \\ E_{k,b}^e(t), \xi_k(t) = 1. \end{cases} \tag{8}$$

2) Communication Model: The model assumes that mobile devices can access BS servers via orthogonal frequency-division multiple access (OFDMA) [29]. OFDMA is a modulation technique. There is orthogonality between subcarriers. It does not interfere with each other by dividing the high-speed data stream into multiple lower-speed subcarriers and transmitting different data on different subcarriers at the same time. This results in more efficient utilization of the channel and better immunity to interference. In addition, interference between devices is ignored when communicating with a single base station. Assuming that there are no obstacles between the devices and BSs, the wireless channel is a line-of-sight (LOS) channel, so the path loss depends on the distance between them. The channel gain between mobile device k and BS b can be expressed as

$$h_{k,b}(t) = \varsigma_0 d_{k,b}^{-\alpha}(t), \tag{9}$$

where ς_0 represents the power gain at 1 m distance.

a) Task Uplink Model: During task offloading, the BS uplink bandwidth B_u is equally allocated to mobile devices with transmission tasks in the communicable area of the BS. The transmission rate between device k and server b is

$$r_{k,b}^u(t) = \frac{B_u}{\mathbb{K}(t)} \log_2\left(1 + \frac{P_k^u h_{k,b}(t)}{\sigma_u^2 + I}\right), \tag{10}$$

where $\mathbb{K}(t)$ is the number of mobile devices within the signal coverage of BS b at time t, P_k^u denotes the task transmit power of mobile device k, σ_u^2 denotes the Gaussian white noise power of each mobile device and I indicates external environmental interference.

Therefore, the transmission time of the mobile device k upload task to BS b can be expressed as

$$T_{k,b}^u(t) = \frac{\partial_k^u(t)}{r_{k,b}^u(t)}, \tag{11}$$

and the transmission energy consumption can be expressed as

$$E_{k,b}^u(t) = P_k T_{k,b}^u(t) = \frac{P_k^u \partial_k^u(t)}{r_{k,b}^u(t)}. \tag{12}$$

b) Task Downlink Model: If, after the task is computed, the mobile device remains within the signal coverage of the BS where the task was computed, i.e., $d'_{k,b}(t) \le R_b$, where $d'_{k,b}(t)$ represents the distance between the mobile device and the computing BS, and R_b denotes the coverage radius of BS b. The task result is transmitted directly to the mobile device via the wireless network. Therefore, the downlink transmission rate is

$$r_{k,b}^r(t) = \frac{B_r}{\mathbb{K}(t)} \log_2\left(1 + \frac{P_k^r h_{k,b}(t)}{\sigma_r^2 + I}\right), \tag{13}$$

where B_r denotes the downlink bandwidth. In this case, the task downlink transmission time is expressed as

$$T_{k,b}^r(t) = \frac{\partial_k^r(t)}{r_{k,b}^r(t)}, \tag{14}$$

and the downlink transmission energy consumption is denoted as

$$E_{k,b}^r(t) = P_k^r T_{k,b}^r(t) = \frac{P_k^r \partial_k^r(t)}{r_{k,b}^r(t)}. \tag{15}$$

If the mobile device leaves the coverage area of the offloading BS after the task computation is completed, i.e., $d'_{k,b}(t) > R_b$, the task computation result needs to be transmitted through a wired network to a reachable BS to complete the downlink transmission of the task. Wired network transmission signals are stable, less susceptible to interference, and have stronger security performance. However, this also incurs additional task migration costs.

A graph is considered for all BSs, where each point represents a BS and the weight of an edge represents the transmission cost between the two connected BSs. It includes the weighted sum of communication delay and transmission energy consumption. The selection of migration paths is implemented by Dijkstra's algorithm. The task migration cost is primarily composed of two elements: delay and energy consumption. The migration cost of transmitting the task result from BS b to w is

$$T_{b,w}^{m}(t) = \sum_{l \in L_{b,w}} \frac{\partial_k^r(t)}{d_l}, \tag{16}$$

where $L_{s,w}$ denotes the set of wired links through which BS s migrates to w, d_l denotes the bandwidth of wired link l, which is related to the distance between BSs.

Therefore, the total transmission delay of the task $T_{k,b}^{trans}(t)$ is expressed as

$$T_{k,b}^{trans}(t) = \begin{cases} T_{k,b}^{u}(t) + T_{k,b}^{r}(t), d'_{k,b}(t) \le R_b, \\ T_{k,b}^{u}(t) + T_{b,w}^{m}(t) + T_{k,b}^{r}(t), d'_{k,b}(t) > R_b, \end{cases} \tag{17}$$

and the total energy consumption of the task $E_{k,b}^{trans}(t)$ is denoted as

$$E_{k,b}^{trans}(t) = \begin{cases} E_{k,b}^{u}(t) + E_{k,b}^{r}(t), d'_{k,b}(t) \le R_b, \\ E_{k,b}^{u}(t) + E_{b,w}^{m}(t) + E_{k,b}^{r}(t), d'_{k,b}(t) > R_b. \end{cases} \tag{18}$$

D. Problem Formulation

The above analysis models task offloading for collaboration among multiple BSs, mainly considering the diversity of task attributes, the heterogeneity of mobile devices and server resources. To minimize the overall system cost, it is crucial to select an effective task offloading strategy and allocate computational resources of the edge servers appropriately. In this model, the task response time is denoted as

$$T_{k,b}^{ttl}(t) = \xi_k^b(t) T_{k,b}^{comp}(t) + \left(1 - \xi_k^b(t)\right)\left(T_{k,b}^{trans}(t) + T_{k,b}^{comp}(t)\right), \tag{19}$$

and the total task energy consumption is expressed as

$$E_{k,b}^{ttl}(t) = \xi_k^b(t) E_{k,b}^{comp}(t) + \left(1 - \xi_k^b(t)\right)\left(E_{k,b}^{trans}(t) + E_{k,b}^{comp}(t)\right), \tag{20}$$

for the task generated by device k in time slot t. The $\xi_k^b(t)$ is the identifier of the location where the task is processed at time t, which is computed locally by the task when $\xi_k^b(t)$ is 1, and is offloaded when 0. The system aims to maximize the system utility, which is representable as

$$U_k(t) = \beta_1(t) \frac{T_k^l(t) - T_{k,b}^{ttl}(t)}{T_k^l(t)} + \beta_2(t) \frac{E_k^l(t) - E_{k,b}^{ttl}(t)}{E_k^l(t)}, \beta_1(t), \beta_2(t) \in (0,1), \tag{21}$$

where the weight factor $\beta_1(t)$ and $\beta_2(t)$ are dynamically adjusted using the entropy weighting method [30]. Additionally, the calculation normalizes $T_k^l(t)$, $T_{k,b}^{ttl}(t)$, $E_k^l(t)$, and $E_{k,b}^{ttl}(t)$. Thus, the optimization problem is expressed as

$$\max_{\xi_k^b(t)} \sum_{t=1}^{T} \sum_{k=1}^{K} U_k(t) \tag{22}$$

$$s.t.\ \xi_k^b(t) \in \{0,1\}, \forall k, b, t \tag{22a}$$

$$\sum_{i=0}^{B} \xi_k^i(t) = 1 \tag{22b}$$

$$C_b(t) \in \{C_c, C_g, C_f\} \tag{22c}$$

$$\partial_k^{su}(t) + T_{k,b}^{ttl}(t) \le \partial_k^{de}(t), \forall k \in K, \forall t \in \mathrm{T} \tag{22d}$$

$$A_k \in \{0,1\}, \forall k \in K \tag{22e}$$

$$\begin{matrix} 0 < P_k^u \le P_{\max}^u \\ 0 < P_k^r \le P_{\max}^r \end{matrix} \tag{22f}$$

$$0 \le y_{C_b(t)} \le Y_{\max}, C_b(t) \in \{C_c, C_g, C_f\}. \tag{22g}$$

Constraint (22a) indicates that a binary offload is used in the system and tasks are not divisible. Constraint (22b) means that a task can be offloaded to only one server. Constraint (22c) denotes that a single task can only select one processor for computation. Constraint (22d) indicates that the task needs to be completed within the deadline. Constraint (22e) indicates that there are two ways for the task to access the BS, including cellular data and WiFi. Constraint (22f) indicates that the task is limited in the transmit power of the end device and the server. Constraint (22g) ensures that the number of computational nodes already processing the task $y_{C_b}(t)$ is less than or equal to the total number Y_{max}.

IV. FedTO Algorithm Details

In this section, first, the overall training framework based on FL in FedTO is introduced. Second, the DRL-based training for each agent is explained. Finally, the algorithm is solved for the pre-deployment of BSs.

A. FL-Based Training Framework

Mobile devices are used for various activities involving increasing amounts of personal information, and the need for privacy protection is becoming stronger. Traditional machine learning training requires data to be collected centrally and trained on a central server. This approach involves a large amount of data transfer and centralized storage, which can easily lead to privacy leakage of users. However, FL decentralizes machine learning algorithms onto local devices. It then uploads training results to a central server via local computation and communication, all without revealing the original data [31]. This approach can effectively protect the privacy of users, while also enabling the use of distributed computing and communication technologies to improve training efficiency and data utilization.

Based on the concept of distributed computing, FL divides the model training process into two phases: local model training and global model aggregation. In the local model training stage, the server sends the current global model to K participants and collects the participants' local model updates $P_l(t)$. Local model updates allow participants to use their own local data to update the model without uploading the data to the server, thus maximizing data privacy. After the participant finishes the model update, it sends the updated model back to the server. In the model aggregation phase, the server side updates the model according to the participants' models. A certain aggregation is used to update the global model $P_g(t)$ to better fit the local data of all participants. FedAvg algorithm in FL is used for model aggregation in this model [32], which is represented as

$$P_g\left(t+1\right) = \frac{1}{K}\sum_{k=1}^{K} P_k(t). \tag{23}$$

B. DRL-Based Offloading Decision

In the first phase described above, the generation process of the offload decision becomes complex because each device has two options to compute a task (i.e., local and edge servers) and the computing power in the system is heterogeneous. As the number of devices and servers increases, the complexity grows exponentially. To address this question, the TD3 algorithm in DRL is applied. The problem is modeled as an MDP, which is described in detail below.

1) MDP Detail Modeling:

The MDP uses a mathematical form to represent the RL problem when the environment is uncertain and stochastic. Define the state space, action space, and reward function to model the environment. Represent this as the transfer probabilities between states and states. The value function is also used to evaluate the merits of state-action pairs.

State Space: At time slot t, when the device sends a request for computing task M_k, the agent collects various information such as the location information of the device, task type, bandwidth resources, and the state and information of all computing nodes in the communicable BSs in the system. The state space of the model is the collection of this information. Thus, the state space S of time slot t is expressed as

$$S = N \times L \times H \times P, \tag{24}$$

where $\times$ denotes the cartesian product.

$$N = \Big\{N_0, \left(N_1^c, N_1^g, N_1^f\right), \ldots, \left(N_i^c, N_i^g, N_i^f\right), \ldots, \\ \times \left(N_s^c, N_s^g, N_s^f\right)\Big\}$$

is the cost of task available for offloading computing nodes in the system. $L = \{L_1, L_2, \ldots, L_i, \ldots, L_B\}$ and $H = \{H_1, H_2, \ldots, H_i, \ldots, H_B\}$ are both B-dimensional sets, which denote the location information of the device and the available bandwidth resource information of the BSs, respectively. P denotes the type of task waiting to be processed. To be specific, N_i is calculated according to

$$N_i = \beta_1(t)T_{k,s}^{ttl}(t) + \beta_2(t)E_{k,s}^{ttl}(t),$$

where N_0 denotes the cost of the task to be calculated locally, N_i^c, N_i^g, N_i^f denote the cost of the CPU, GPU, and FPGA in the BS server i to process the task, respectively. $L_i \in \{0, 1\}$, $L_i = 1$ means that the device generating the task is within the signal coverage of the i-th BS and can perform the task; $L_i = 0$ means that it is not within the range and cannot perform the task. H_i denotes the available bandwidth resource of the edge BS, which depends on the type of wireless network it deploys and the number of mobile devices within the range of the BS at time t.

Action Space: The agent in the mobile device decides which computing node of which BS to offload the task to based on the state of the environment. The set of actions A represents all the selectable actions and can be represented as

$$A = \left(a_0, a_1^c, a_1^g, a_1^f, \ldots, a_i^c, a_i^g, a_i^f, \ldots, a_s^c, a_s^g, a_s^f\right). \tag{25}$$

A is a set of one-hot vector, where $a_0, a_i^c, a_i^g, a_i^f \in \{0, 1\}$. $a_0 = 1$ means that the task is executed locally. If the task is offloaded to the CPU processor of the 1st BS for execution, then $A = (0, 1, 0, 0, \ldots, 0, 0, 0)$.

Reward Function: The reward function is denoted by $R(t)$, which describes the reward an agent receives after performing an action a in state s. It is used to guide the agent to learn the appropriate policy. The aim of this model is to minimize the response time of the task and energy consumption to maximize the system utility. The $R(t)$ is determined as a negative weighted sum of the response time and energy consumption when all constraints in (21) are satisfied, which can be expressed as $Cost_{k,s}(t) = -(\beta_1(t)T_{k,s}^{ttl}(t) + \beta_2(t)E_{k,s}^{ttl}(t))$. Otherwise, the value of the reward function is the corresponding penalty of -1. which can be expressed as

$$R(t) = \begin{cases} Cost_{k,s}(t), \text{if satisfying constraints}, \\ -1, \text{otherwise}. \end{cases} \tag{26}$$

Therefore, $R(t) \in [-1, 0)$. The equation demonstrates that a better task offloading strategy results in a smaller system cost and a higher value for the reward function.

2) Classical Q-Learning Algorithm: The Q-learning algorithm is a well-known RL algorithm that is based on the value function. Its main objective is to learn the optimal policy by iteratively updating the Q-values [33]. The update formula of Q-learning is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right). \tag{27}$$

The learning rate is denoted by α, and the immediate payoff obtained by the agent performing an action a in state s is represented by r. γ is the discount factor, and $\max_{a'} Q(s', a')$ is the maximum Q-value obtained by taking the optimal strategy in state s'. The Q-Learning algorithm is illustrated in Algorithm 1, with an algorithmic complexity of $O(ET)$, where E denotes the number of episodes and T denotes the size of the total timestamp.

First, the initial action-value function $Q(s, a)$ is randomly set and the action a is selected using the ϵ-greedy strategy based on the current state s and the Q-function. ϵ-greedy is a common explore-exploit strategy used in RL. At each time step, the agent takes a random action with ϵ probability (explore) and chooses the action with the highest estimated reward with $1-\epsilon$ probability (exploit). Then, the selected action a is executed and feedback from the environment is obtained, including the reward r and the next state s'. The Q-function is updated and the next state s' is assigned to the current state s. Finally, the

Algorithm 1: The Q-Learning Algorithm.

```
Input: The sets related to the communication between
       local devices and BS servers.
Output: The optimal action-value function $Q^*(s,a)$
1  Initialize $Q(s,a)$ randomly for all $s \in S, a \in A$;
2  for each episode do
3      for $i = 1$ to $T$ do
4          The agent obtains the status $s$ of the current
             environment;
5          With probability $\epsilon$ choose a random action $a$,
             otherwise choose $a = \arg\max_{a'} Q(s,a')$;
6          Take action $a$ and observe reward $r$ and next
             state $s'$;
7          Update according to Eq. (27);
8          $s \leftarrow s'$;
9      end
10 end
11 return $Q^*(s,a)$
```

Algorithm 2: TD3-Based Task Offloading Algorithm.

```
Input: Mobile device trajectory data, system state,
       channel state and task state
Output: Task average response time and system total
        energy consumption
1  Initialize the actor network $\theta^v$ and critic networks
     $\theta^{Q_1}, \theta^{Q_2}$.
2  Initialize target networks $\theta^{v'} \leftarrow \theta^v$, $\theta^{Q_1'} \leftarrow \theta^{Q_1}$,
     $\theta^{Q_2'} \leftarrow \theta^{Q_2}$
3  Initialize experience pool $R$
4  for $t$ = 1, $T$ do
5      The offloading decision vector is obtained from the
         actor network, according to
         $a(t) = v(s(t)|\theta^v) + \varsigma$.
6      In state $s(t)$, the reward $r(t)$ and the next moment
         state $s(t+1)$ are obtained, according to the
         action $a(t)$.
7      Save the record $\langle s(t), a(t), r(t), s(t+1)\rangle$ in $R$.
8      Sample a mini-batch of $Z$ records from $R$;
9      Calculate the target Q-value according to Eq. (28),
         (29);
10     Update the critic network according to Eq. (30),
         (31);
11     if $t$ mod $d$ then
12         Update the actor network parameters according
             to Eq. (32);
13         Soft update target networks according to Eq.
             (33), (34);
14     end
15 end
16 return
```

above steps are repeated until the Q-function converges and the optimal action policy is output.

However, Q-learning has the following weaknesses. 1) It is not suitable for continuous states and action spaces, but only for discrete cases. 2) It is only applicable to the case of a single objective. In the specific realization, it is often necessary to consider multiple objectives at the same time, such as response time and energy consumption. 3) When the state space is too large, the Q-learning algorithm needs to save all the Q-value functions, resulting in a large computational and storage overhead, which makes it difficult to apply to issues with massive state spaces. The main algorithm to address the above problem is DQN [34]. However, since the Q-value is used as an estimate of action value in the DQN algorithm, the problem of overestimation may arise if the training data of the neural network is insufficient or unevenly sampled. In addition, it also cannot handle continuous action space and tends to suffer from training instability. Therefore, we combine FL and implement the TD3 algorithm into the multiple BSs collaborative task offloading.

3) TD3 Algorithm: The model training and optimization in the TD3 algorithm is mainly performed through the actor and critic networks [35]. Both the training of actor and critic rely on the reward signals received from the environment, but they have different optimization goals: the actor's goal is to maximize the value function, i.e., to maximize the long-term expected reward, while the critic's goal is to minimize the gap between the value function and the true reward.

In the TD3 algorithm, the main role of the actor network is to map the current state s to an optimal action a. It consists of several fully connected layers, where each input module receives different information about the current state, i.e., the trajectory data of the mobile device, the channel state, the MEC system state and the task state. The fully connected layers in the input modules transform this information into a set of features. Next, these features go through the connection layer for feature fusion to generate the optimal action. Finally, the actor network outputs the action probabilities through its output layer. The TD3 algorithm contains two actor networks, $v(s|\theta^v)$ is used to represent the actor network and $v'(s|\theta^{v'})$ is used to represent the target actor network. The main function of the critic network is to estimate the value function (i.e., the expected cumulative reward) of the current policy for a given state, and its inputs include the current system state and actions. The state includes the trajectory data of the mobile device and the cost of the system, while the actions are the task offloading vectors output by the actor network. Similar to the actor network, the critic network also consists of multiple fully connected layers and connects their outputs together to be sent to the feature fusion module. Finally, the critic network outputs an estimate of the current state and the Q-value of the action, which is used to update the policy of the actor network. There are four critic networks in TD3, and two critic networks are denoted by $Q_1(s,a|\theta^{Q_1})$ and $Q_2(s,a|\theta^{Q_2})$, respectively, and $Q'_1(s,a|\theta^{Q'_1})$ and $Q'_2(s,a|\theta^{Q'_2})$ to denote two target critic networks, respectively.

The policy network in the Q-based RL algorithm performs an update of the parameters by maximizing the target value y, which can be expressed as $y = r + \gamma \max_{a'} Q(s',a')$. However, this target is likely to be affected by sample error, resulting in a maximum value of the action value estimate that is usually larger than

Algorithm 3: K-Means-Based Algorithm for Pre-Deployment of BSs Locations.

```
   Input: All trajectory data of the user at a specific
          time, the number of BSs in the MEC system $B$
   Output: Coordinates of $B$ BSs
1  Uniformly initialize $B$ locations as the initial BSs
     placement locations;
2  while not converge do
3      for each $k \in K$ do
4          Compute the distance from
             $o_k = (x_k(t), y_k(t))$ to all BSs in $B$ and
             divide them into $B$ clusters.
5          Allocate device $k$ to the closest cluster
             $c_j = (c_j^x, c_j^y)$:
               $S_j = \{x_i : ||x_i - c_j^x|| \leq ||x_i - c_l^x||,$
                  $y_i : ||y_i - c_j^y|| \leq ||y_i - c_l^y||, \forall l \neq j\}$
6      end
7      Update each centroid $c_j$ as the mean of all data
         points in its cluster: $c_j = \frac{1}{|S_j|}\sum_{x_i,y_i \in S_j}(x_i, y_i)$;
8      if the cluster assignments and centroids do not
         change then
9          return Coordinates of $B$ BSs
10     end
11 end
```

the true value, i.e., $\mathrm{E}_\varsigma[\max_{a'}(Q(s',a')+\varsigma)] \geq \max_{a'}Q(s',a')$. This error propagates through the Bellman equation and for this problem, a double Q-learning strategy is used in TD3. The number of critic networks used to estimate Q-values is twice the number of deep deterministic policy gradients (DDPG) [36]. The target Q-value is taken as the smaller value of $Q'_1(s,a|\theta^{Q'_1})$ and $Q'_2(s,a|\theta^{Q'_2})$. Target Q-value is calculated by

$$\hat{a} = v'\left(s(t+1)|\theta^{v'}\right) + \varsigma, \tag{28}$$

$$Q = r(t) + \gamma \min_{i=1,2} Q'_i\left(s(t+1), \hat{a}|\theta_i^{Q'}\right), \tag{29}$$

where ς is the noise introduced, following a Gaussian distribution. In the training phase, a batch of data $\langle s(t), a(t), r(t), s(t+1)\rangle$ are sampled from the experience pool. For the critic networks, the error loss between the estimate and target value is minimized using the gradient descent algorithm, which leads to the update of $Q_1(s,a|\theta^{Q_1})$ and $Q_2(s,a|\theta^{Q_2})$, denoted as

$$\mathrm{Loss}_{Q_1} = \frac{1}{Z}\sum_{i=1}^{Z}\left(\hat{Q} - Q_1\left(s,a|\theta^{Q_1}\right)\right)^2, \tag{30}$$

$$\mathrm{Loss}_{Q_2} = \frac{1}{Z}\sum_{i=1}^{Z}\left(\hat{Q} - Q_2\left(s,a|\theta^{Q_2}\right)\right)^2, \tag{31}$$

where Z is the size of a batch sample. After the critic networks are updated d steps, the update of the actor network is activated. For the actor network, the gradient ascent method is used, denoted as

$$\nabla_{\theta^o} J \approx \frac{1}{M}\sum_i \nabla_a Q\left(s,a|\theta^Q\right)|_{s=s_i, a=o(s_i)} \nabla_{\theta^o} o\left(s|\theta^o\right)|_{s_i}. \tag{32}$$

All target networks use soft updating to ensure the stability of the algorithm, denoted as

$$\theta^{v'} = \tau\theta^v + (1-\tau)\theta^{v'}, \tag{33}$$

$$\theta^{Q'_i} = \tau\theta^{Q_i} + (1-\tau)\theta^{Q'_i}\,(i=1,2), \tag{34}$$

where τ denotes the soft update rate, $\tau \in [0,1]$. The pseudo-code of TD3 algorithm is shown in Algorithm 2 with a time complexity of $O(T(m+n))$.

C. K-Means-Based Pre-Deployment for BSs

When initializing the system, the location of the edge BSs needs to be initialized. In this model, a k-means based data-driven mobile BSs pre-deployment algorithm is shown in Algorithm 3, with a time complexity of $O(Bmn)$, where B is the number of base stations, i.e., the number of clusters. m and n are the number of iterations and the number of training samples, respectively. Specifically, the algorithm takes the trajectory data of mobile devices for a known period of time as input and clusters the trajectory data into B clusters using the k-means algorithm. For each cluster, the average of all its trajectory data points is calculated as the center of mass of that cluster. Each center of mass is initialized as the location of an edge BS. Provide services to subsequent unknown users in the region to reduce the cost of performing the task.

Based on the above analysis for the FL training framework, TD3 offloading decision algorithm and BS pre-deployment algorithm, the FedTO algorithm is summarized in Algorithm 4. The algorithm complexity is $O(T^2K(m+n))$.

V. Experiment Results

In this section, first, a mobile device trajectory dataset in a real environment is presented. Then, the proposed algorithm FedTO is compared with other RL algorithms, DDPG, DDQN (double deep Q-Network), and DQN (deep Q-Network), in the framework of FL.

A. Trajectory Dataset

In previous task offloading models, theoretical assumptions or artificially generated trajectory data are usually used to conduct simulation experiments. Although this approach can explore the performance of task offloading strategies to a certain extent, it may lead to biased evaluation of system performance due to the discrepancy between generated and real data. To more accurately simulate users' movement patterns and behaviors, and thus better design task unloading strategies, this model uses real user data instead of virtual trajectory data. This dataset is part of the GPS track dataset used by the Microsoft Asia Research Institute GeoLife project, which collected trajectory data from 182 users from April 2007 to August 2012 [37]. It represents a chronological collection of sites, each containing

Algorithm 4: The FedTO Algorithm.

```
   Input: Task set and device states
   Output: Task average response time and system total
           energy consumption
1  Initialize the edge BSs locations according to
     Algorithm 3.
2  Edge BS side: Initialize the global model with
     random parameter values $P_g(0)$ at time $t = 0$;
3  End device side: Download $P_g(0)$ from the edge BS
     server and let $P_k(0) = P_g(0)$, $(k = 1, 2, \cdots, K)$;
4  for $t$ = 1, $T$ do
5      End device side:
6      for each end devive $k \in K$ in parallel do
7          Download $P_g(t)$ from edge;
8          Let $P_k(t) = P_g(t)$;
9          for each task generated by device $k$ do
10             Select a suitable computing node for the
                 task based on $P_k(t)$ with Algorithm 2;
11             Train the local model $P_k(t)$;
12         end
13         Upload the trained model parameters $P_k(t+1)$
             to the edge;
14     end
15     Edge server side:
16     Receive local models of all devices;
17     Use the federated averaging to construct global
         model $P_g(t+1)$ according to
         $P_g(t+1) = \frac{1}{K}\sum_{k=1}^{K} P_k(t)$;
18     Distribute the global model;
19 end
20 return
```

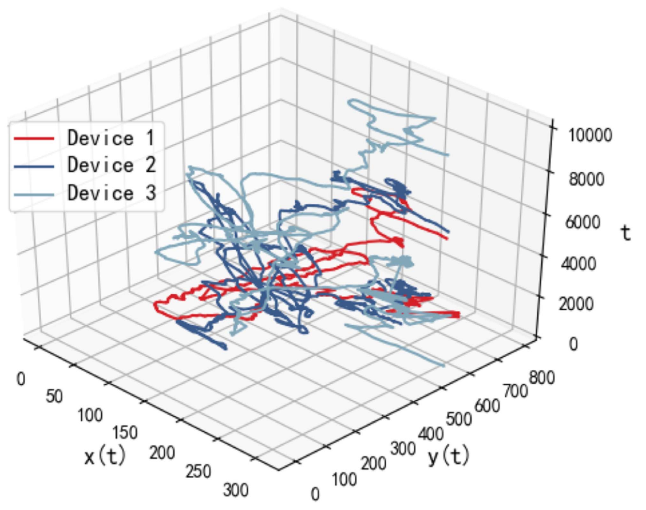


Fig. 2. Movement trajectory of different users.

TABLE III
EXPERIMENTAL SIMULATION PARAMETERS

Parameters	Value
I (dBm)	-90
P_k^u (mW)	100
P_k^r (mW)	200
T (s)	10^4
f_k (GHz)	Unif(3.0, 4.0)
f_s^{ty} (C_c) (GHz)	[Unif(9, 10), Unif(8, 9), Unif(7, 8), Unif(7, 8)]
f_s^{ty} (C_g) (GHz)	[Unif(9, 9.5), Unif(9.5, 10), Unif(8, 9), Unif(7, 8)]
f_s^{ty} (C_c) (GHz)	[Unif(8, 10), Unif(8, 10), Unif(8, 9), Unif(8, 9)]
σ_r^2, σ_u^2 (dBm)	-174
ς_0 (dBm)	-50
κ	10^{-28}

information about the latitude, longitude, altitude, speed, and current direction. These trajectory data are collected by different GPS devices with a frequency of one point every 2–5 seconds or every 5–10 meters. In the following experiments, the pedestrian trajectories are filtered out, and the WGS84 coordinates in the dataset are transformed into a transcendental Gaussian coordinate representation. The trajectories of three different users over 10000 seconds are shown in Fig. 2, which illustrates the movement features of the users. Fig. 3 shows the aggregation of all users at 0 seconds, 5000 seconds, and 10000 seconds, which illustrates a significant aggregation in the location distribution of the devices. In the entire process, each user uniformly generates tasks at a specific time, bounded by the task density. The size of the tasks is exponentially distributed around a specified value. The types of tasks are randomly distributed across devices at different scales.

B. Experimental Settings

The FedTO algorithm is implemented using the Python and the TensorFlow framework. In addition, the Cloudsim platform was used to simulate the entire process of task generation and offloading in the system. Based on a custom network topology, computation and communication resources are scaled and task offloading strategy is customized. In the experiments, pre-deployment edge BSs are located, the tasks of mobile devices are generated uniformly according to a certain density, and the task size is exponentially distributed over a range. The main parameters are listed in the Table III [38] [39]. The other RL algorithms in the experiment are described as follows:

1) DQN (Deep Q-Network): DQN is an RL algorithm that approximates the Q-value function using deep neural networks. It improves training stability through empirical playback and fixed target networks.
2) DDQN (Double Deep Q-Network): DDQN is an improvement of traditional DQN. DDQN introduces two Q networks, one for selecting actions and the other for evaluating the value of actions, thus improving training stability and mitigating the overestimation problem.
3) DDPG (Deep Deterministic Policy Gradient): DDPG is used to solve continuous action space problems. It combines the policy gradient method and deep Q-network ideas to optimize the policy for continuous actions by approximating the Q-value and the policy function. It

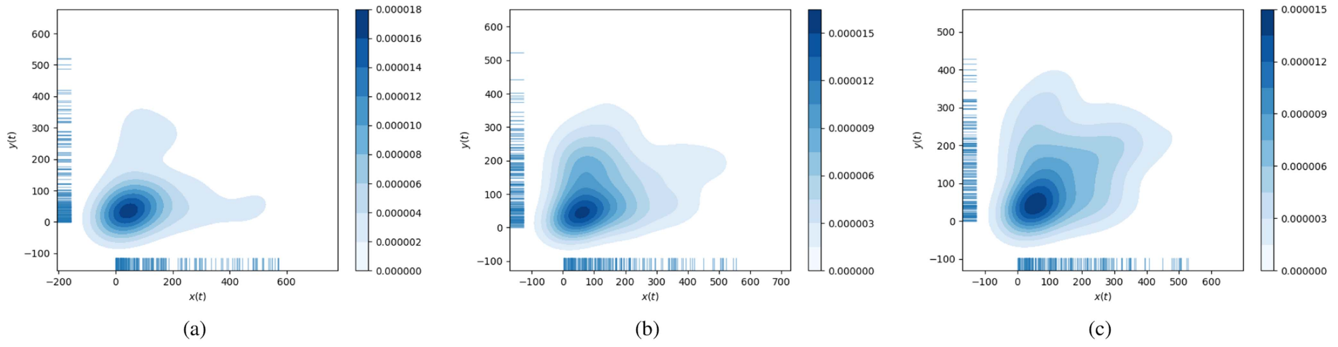


Fig. 3. Distribution density of user devices at different moments. (a) At 0 seconds, (b) At 5000 seconds, (c) At 10000 seconds.

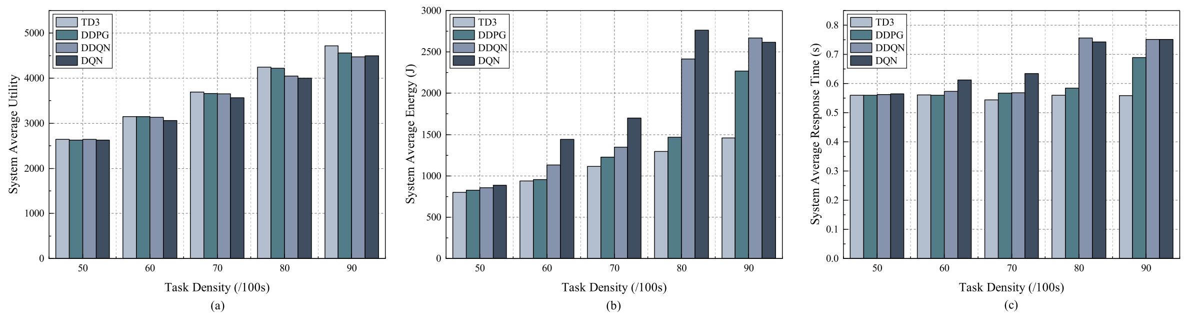


Fig. 4. Performance comparison at various task densities. (a) System utility. (b) Total energy consumption. (c) Average response time.

performs well in exploring continuous action space and stable training.

Similar to the TD3 algorithm applied in FedTO, all of the above belong to the field of DRL, which uses deep neural networks to approximate the value function or policy function. However, there are differences in objectives, scope of application, network structure and strategies.

C. Task Density

In this part of the experiment, two system cost metrics and utilities are compared for four RL algorithms with different task generation densities. The task densities are 50, 60, 70, 80, and 90 tasks generated by a single user in 100 seconds. The 3 mobile users in the system are considered. The experimental results are shown in Fig. 4.

The experimental results show that the system utility is optimized by an average of 2.25%. At different task densities, the performance of different RL algorithms in the FL framework differs. With increased task density, the system utility, energy consumption, and average response time all increase. This is because at high task densities, the number of tasks increases and the computational and communication resources in the system are fixed, requiring more energy and time to execute the tasks. The increase in energy consumption, latency of individual tasks and the total number of tasks directly leads to the increase in the total energy consumption and average response time of the system. All RL algorithms perform relatively well under low-density tasks, but the performance of DQN and DDQN algorithms decreases rapidly with increasing task density. The TD3 algorithm has higher system utility with lower energy consumption and response time compared to other algorithms.

TABLE IV
TASK SIZE IN DIFFERENT SCENARIOS (MB)

Scenario	Text	Image	Audio	Video
I	0.25	0.5	0.75	1
II	0.5	1	1.5	2
III	0.75	1.5	2.25	3
IV	1	2	3	4

D. Task Size

In this section of the experiments, three performance metrics are compared in the case of different task generation sizes. There are 3 mobile users in the system. The task density is 100s generating 70 tasks, and the combinations of upload task sizes are shown in Table IV. The experimental results are shown in Fig. 5.

The experimental results show that the system utility is optimized by an average of 1.5% compared to other RL algorithms. Moreover, the performance of the RL algorithm varies greatly depending on the task sizes. With an increasing size of the task, the system utility shows a stable trend, but the total system energy consumption and the average response time of the tasks increase sharply. This is because the larger the task is, the more energy is required to transfer the task to the edge BS and the longer the execution time will be. When the resources in the system are fixed, there will be more tasks waiting to be computed at the same moment. As the task size increases,

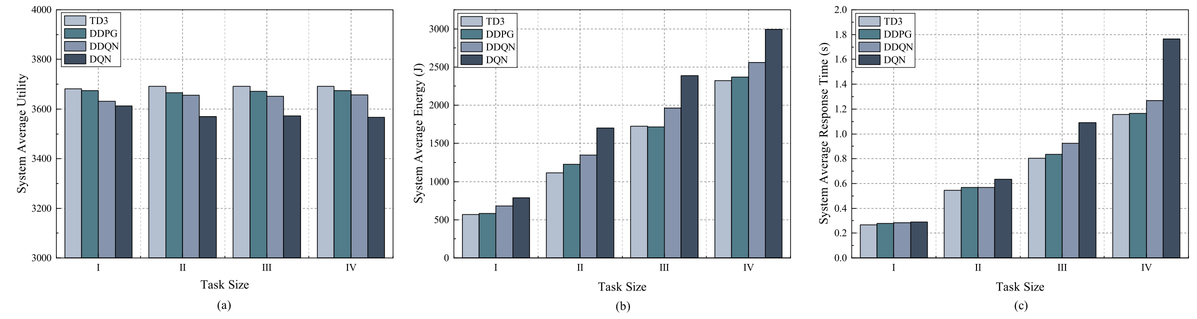


Fig. 5. Performance comparison at various task sizes. (a) System utility. (b) Total energy consumption. (c) Average response time.

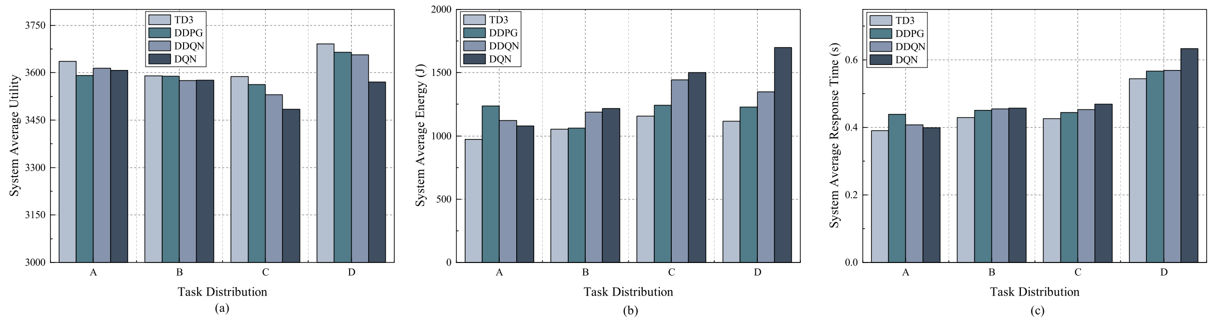


Fig. 6. Performance comparison at various task distributions. (a) System utility. (b) Total energy consumption. (c) Average response time.

TABLE V
TASK DISTRIBUTION IN EACH DEVICE

Scenario	Sizes proportion	Types proportion
A	Same	Different
B	Different	Same
C	Different	Different
D	Same	Same

the performance of the DDPG, DDQN, and DQN algorithms deteriorates faster relative to TD3, resulting in algorithms that exhibit unstable and suboptimal performance. Specifically, the DQN algorithm performs the worst at all densities, while the TD3 algorithm performs the best at all densities. The DDQN algorithm performs better at low densities, but its performance decreases significantly at high densities. This is because the TD3 and DDPG algorithms use continuous action space and neural network function approximation methods. This allows for better handling complex states and action spaces, and better adaptation to multitasking scenarios. The DQN algorithm suffers from overestimation of the Q-value in exploration.

E. Task Distribution

In this part of the experiment, different performance metrics of four RL algorithms are compared under different task assignment rules. The task density is 70 per 100 seconds for a single user. The distribution of tasks in the experiment is shown in Table V. The results are displayed in Fig. 6.

Compared with other algorithms, the TD3 algorithm improves the system utility by 1.14% on average, but the performance varies under different task distribution rules. The system utility is highest when the task type and size distribution are consistent, and lowest when both task distribution and task size distribution are inconsistent, with the DQN algorithm showing a significant decrease in performance. Therefore, the variation of task distribution and task size distribution affects the variation of task response time and energy consumption.

F. Bandwidth

In this section, the performance of four different RL algorithms is considered to be evaluated at different channel bandwidth sizes, i.e., 15 MHz, 25 MHz, 35 MHz, and 45 MHz. The task density is 70 by a single user in 100 seconds. The experimental results are shown in Fig. 7.

The experimental results demonstrate that the system utility is optimized by 2.4% on average compared to other RL algorithms, and the performance of the RL algorithm varies with the channel bandwidth size. As the bandwidth gradually increases, the system utility gradually increases, and the task response time and total energy consumption both decrease accordingly. When the bandwidth is small, the communication speed between nodes is limited, so it takes longer to complete the task transmission. This leads to a longer task response time and increases energy consumption. In addition, the slower task transmission speed increases the communication overhead and the system utility decreases. When the maximum channel bandwidth size is 45 MHz, the differences in performance metrics are low for all four RL algorithms. However, when the channel bandwidth size is reduced to 15 MHz, TD3 and DDPG algorithms significantly outperform DDQN and DQN. When the channel bandwidth size is further increased to 25 MHz and 35 MHz, TD3 and DDPG still achieve better performance than DDQN and DQN in all three

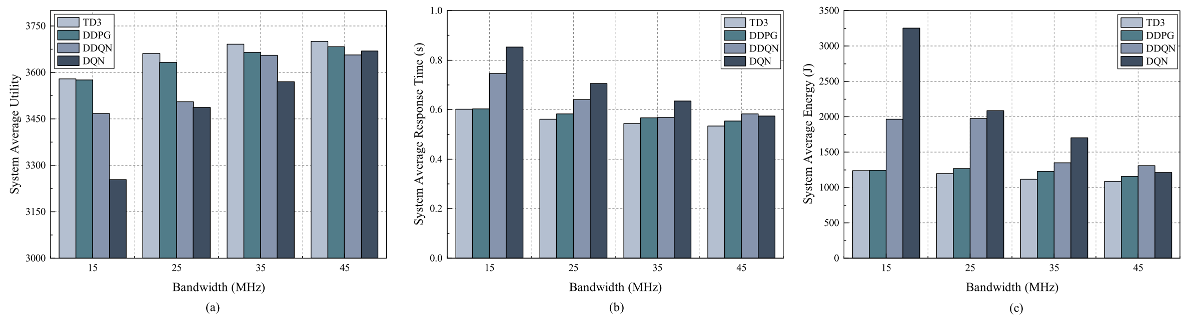


Fig. 7. Performance comparison at various bandwidths. (a) System utility. (b) Total energy consumption. (c) Average response time.

aspects. This is due to the use of dual-Q networks and delayed updates in TD3, which better solve the action value estimation bias problem and make TD3 have better learning performance.

VI. Conclusion

In this study, a mobile user task offloading problem in a multiple BSs collaborative MEC system is considered. The main goal of the system is to safeguard user privacy while trade-off optimizing energy consumption and task response time, which is modeled as an MDP. A task offloading algorithm FedTO based on DRL in FL framework is proposed. The algorithm dynamically selects the appropriate compute node to perform the task based on the real-time network state and the location information of the mobile device. Experiments are conducted with the real mobile device trajectories, and it is shown that the algorithm optimizes the system utility and achieves the purpose of reducing energy consumption and task response time while ensuring task safety, making the task offloading process safer and more efficient. In future work, we will study more optimization objectives of MEC and consider optimization problems for FL aggregation in large-scale systems and apply dynamic bandwidth to the model.

Acknowledgment

The authors thank the editors and reviewers for their insightful comment and valuable suggestions.

References

[1] B. B. Sinha and R. Dhanalakshmi, “Recent advancements and challenges of internet of things in smart agriculture: A survey,” *Future Gener. Comput. Syst.*, vol. 126, no. 4, pp. 169–184, 2022.

[2] M. Hasan, “State of IoT – spring 2022,” 2022. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[3] K. Lone and S. A. Sofi, “A review on offloading in fog-based internet of things: Architecture, machine learning approaches, and open issues,” *High-Conf. Comput.*, vol. 3, 2023, Art. no. 100124.

[4] M. Y. Akhlaqi and Z. B. Mohd Hanapi, “Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions,” *J. Netw. Comput. Appl.*, vol. 212, 2023, Art. no. 103568.

[5] Z. Sun et al., “Cloud-edge collaboration in industrial Internet of Things: A joint offloading scheme based on resource prediction,” *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17014–17025, Sep. 2022.

[6] L. Tan, Z. Kuang, L. Zhao, and A. Liu, “Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing,” *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022.

[7] Y. Kang et al., “HWOA: An intelligent hybrid whale optimization algorithm for multi-objective task selection strategy in edge cloud computing system,” *World Wide Web*, vol. 25, no. 5, pp. 2265–2295, 2022.

[8] J. Huang, M. Wang, Y. Wu, Y. Chen, and X. Shen, “Distributed offloading in overlapping areas of mobile-edge computing for Internet of Things,” *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13837–13847, Aug. 2022.

[9] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, and A. S. Avestimehr, “Federated learning for the Internet of Things: Applications, challenges, and opportunities,” *IEEE Internet Things Mag.*, vol. 5, no. 1, pp. 24–29, Mar. 2022.

[10] M. Tang and V. W. S. Wong, “Deep reinforcement learning for task offloading in mobile edge computing systems,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[11] 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.

[12] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, “Deep reinforcement learning for mobile edge caching: Review, new features, and open issues,” *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov./Dec. 2018.

[13] X. Dai et al., “Task co-offloading for D2D-assisted mobile edge computing in industrial internet of things,” *IEEE Trans. Ind. Inform.*, vol. 19, no. 1, pp. 480–490, Jan. 2023.

[14] S. Yue et al., “Todg: Distributed task offloading with delay guarantees for edge computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1650–1665, Jul. 2022.

[15] K. Gai, X. Qin, and L. Zhu, “An energy-aware high performance task allocation strategy in heterogeneous fog computing environments,” *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 626–639, Apr. 2021.

[16] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, “EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing,” *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.

[17] Y. Liu, L. Jiang, Q. Qi, and S. Xie, “Energy-efficient space-air-ground integrated edge computing for internet of remote things: A federated DRL approach,” *IEEE Internet Things J.*, vol. 10, no. 6, pp. 4845–4856, Mar. 2023.

[18] Z. Tong, J. Wang, Y. Wang, B. Liu, and Q. Li, “Energy and performance-efficient dynamic consolidate VMS using deep-Q neural network,” *IEEE Trans. Ind. Inform.*, vol. 19, no. 11, pp. 11030–11040, Nov. 2023.

[19] L. Ai, B. Tan, J. Zhang, R. Wang, and J. Wu, “Dynamic offloading strategy for delay-sensitive task in mobile-edge computing networks,” *IEEE Internet Things J.*, vol. 10, no. 1, pp. 526–538, Jan. 2023.

[20] X. Yao, N. Chen, X. Yuan, and P. Ou, “Performance optimization of serverless edge computing function offloading based on deep reinforcement learning,” *Future Gener. Comput. Syst.*, vol. 139, pp. 74–86, 2023.

[21] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, “Accelerating deep learning inference via model parallelism and partial computation offloading,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 475–488, Feb. 2023.

[22] T. Li, X. He, S. Jiang, and J. Liu, “A survey of privacy-preserving offloading methods in mobile-edge computing,” *J. Netw. Comput. Appl.*, vol. 203, 2022, Art. no. 103395.

[23] D. Wei, N. Xi, X. Ma, M. Shojafar, S. Kumari, and J. Ma, “Personalized privacy-aware task offloading for edge-cloud-assisted industrial internet of things in automated manufacturing,” *IEEE Trans. Ind. Inform.*, vol. 18, no. 11, pp. 7935–7945, Nov. 2022.

[24] J. Zhou, T. Wang, W. Jiang, H. Chai, and Z. Wu, “Decomposed task scheduling for security-critical mobile cyber–physical systems,” *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22280–22290, Nov. 2022.

[25] S. Wang, J. Li, G. Wu, H. Chen, and S. Sun, "Joint optimization of task offloading and resource allocation based on differential privacy in vehicular edge computing," *IEEE Trans. Comput. Social Syst.*, vol. 9, no. 1, pp. 109–119, Feb. 2022.

[26] G. Zhang, S. Ni, and P. Zhao, "Learning-based joint optimization of energy delay and privacy in multiple-user edge-cloud collaboration MEC systems," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1491–1502, Jan. 2022.

[27] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2622–2629, Apr. 2020.

[28] C. Chen, K. Li, A. Ouyang, Z. Zeng, and K. Li, "Gflink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1275–1288, Jun. 2018.

[29] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12313–12325, Dec. 2018.

[30] Z. Tong, J. Wang, J. Mei, K. Li, W. Li, and K. Li, "Multi-type task offloading for wireless internet of things by federated deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 145, pp. 536–549, 2023.

[31] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*.

[32] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," 2019, *arXiv:1907.02189*.

[33] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annu. Rev. Statist. Appl.*, vol. 7, pp. 279–301, 2020.

[34] V. Mnih et al., "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[35] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.

[36] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning,"2015, *arXiv:1509.02971*.

[37] Y. Zheng et al., "GeoLife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

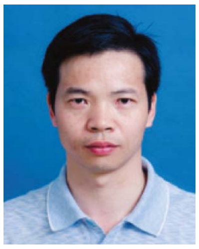
[38] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.

[39] Z. Tong, B. Liu, J. Mei, J. Wang, W. Li, and K. Li, "D2op: A fair dual-objective weighted scheduling scheme in internet of everything," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 9206–9219, May 2023.

Zhao Tong (Member, IEEE) received the Ph.D degree in computer science from Hunan University, Changsha, China in 2014. He was a visiting scholar with the Georgia State University, Atlanta, GA, USA, from 2017 to 2018. He is currently an Associate Professor with the College of Information Science and Engineering, Hunan Normal University, Changsha, China and the Young Backbone Teacher of Hunan Province, China. His research interests include parallel and distributed computing systems, resource management, Big Data, and machine learning algorithm. He has published more than 25 research papers in international conferences and journals, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *Information Sciences*, *Future Generation Computer Systems*, *Neural Computing and Applications*, *Journal of Parallel and Distributed Computing*, and PDCAT. He is a Senior Member of the China Computer Federation.

Jiake Wang received the BE degree from the Hunan Institute of Science and Technology, Yueyang, China in 2021. She is currently working toward the M.S. degree with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests include mobile edge computing, deep reinforcement learning algorithms, and federated learning.

Jing Mei received the Ph.D. degree in computer science from Hunan University, Yueyang, China, in 2015. She is currently an Assistant Professor with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests include parallel and distributed computing and cloud computing. She has published 12 research articles in international conference and journals, such as IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON SERVICE COMPUTING, *Cluster Computing*, *Journal of Grid Computing*, and *Journal of Supercomputing*.



Kenli Li (Senior Member, IEEE) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2003. He was a Visiting Scholar with the University of Illinois, Urbana-Champaign, IL, USA, from 2004 to 2005. He is currently a full Professor of computer science and technology with Hunan University and the Deputy Director of National Supercomputing Center, Changsha, China. His research interests include parallel computing, cloud computing, and Big Data computing. He has published more than 300 papers in international conferences and journals. He serves on the editorial boards of IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE Transactions on Sustainable Computing, *International Journal of Pattern Recognition*, and *Artificial Intelligence*. He is an outstanding Member of CCF.

Keqin Li (Fellow, IEEE) is currently a SUNY Distinguished Professor of computer science. He has authored or coauthored more than 510 journal articles, book chapters, and refereed conference papers. His research interests include parallel computing and high-performance computing, distriuted computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-hysical systems. He was the recipient of the several best paper awards. He is on the Editorial boards of the IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, and IEEE Transactions on Sustainable Computing.